HELPER-X: A UNIFIED INSTRUCTABLE EMBODIED AGENT TO TACKLE FOUR INTERACTIVE VISION-LANGUAGE DOMAINS WITH MEMORY-AUGMENTED LANGUAGE MODELS

Gabriel Sarch Sahil Somani* Raghav Kapoor*
Michael J. Tarr Katerina Fragkiadaki

*equal contribution

Carnegie Mellon University

helper-agent-llm.github.io

ABSTRACT

Methods for developing instructable embodied artificial agents typically train distinct models for each application and language domain to map instructions to the corresponding actions and task plans. Here we explore the feasibility of developing a versatile "generalist" instructable agent capable of operating across a broad spectrum of tasks, language domains, and environments, with a single model. Recent research on instructable agents has used memory-augmented Large Language Models (LLMs) as task planners, a technique that retrieves language-program examples relevant to the input instruction and uses them as in-context examples in the LLM prompt to improve the performance of the LLM in inferring the correct action and task plans. Our approach, HELPER-X, expands such external language-program memory with a wide range of examples and prompt templates, while also extending the agent's action API. This expansion of a shared unified memory enables the agent to work across the domains of executing plans from dialogue, natural language instruction following, active question asking, and commonsense room reorganization. We evaluate the agent on four diverse interactive visual-language embodied agent benchmarks: ALFRED, TEACh, DialFRED, and the Tidy Task. These benchmarks vary significantly in terms of input instructions, question-asking capabilities, task structures, and environmental settings. HELPER-X achieves few-shot, state-of-the-art performance across these benchmarks using a single agent, without requiring in-domain training, and remains competitive with agents that have undergone in-domain training. Our work demonstrates the potential of memory-augmented large language models to support generalist instructable embodied agents.

1 INTRODUCTION

AI is undergoing a paradigm shift with the emergence of models like CLIP (Radford, 2021), GPT (gpt, 2023) and DALL-E (Betker, 2023), which are trained using web-scale data and are adaptable to a wide range of downstream tasks. Adapting vision-language models (VLMs) (e.g., CLIP Radford (2021)) and large language models (LLMs) (e.g., GPT-4 OpenAI (2023)) trained on large datasets often outperform their counterparts trained on smaller scale, task-specific data (Radford, 2021; OpenAI, 2023).

A typical way to adapt LLMs to downstream applications is through prompting (Brown, 2020; Alayrac, 2022; Liu, 2022b; Hongjin, 2022; Mishra, 2022; Wei, 2021; Song, 2022b), exploiting their strong in-context and few-shot learning abilities. Prompting freezes the model parameters and steers the output of the model by conditioning on a prompt – input text typically composed of task instructions and demonstrations. To ensure high performance on a given task, prompts need to be

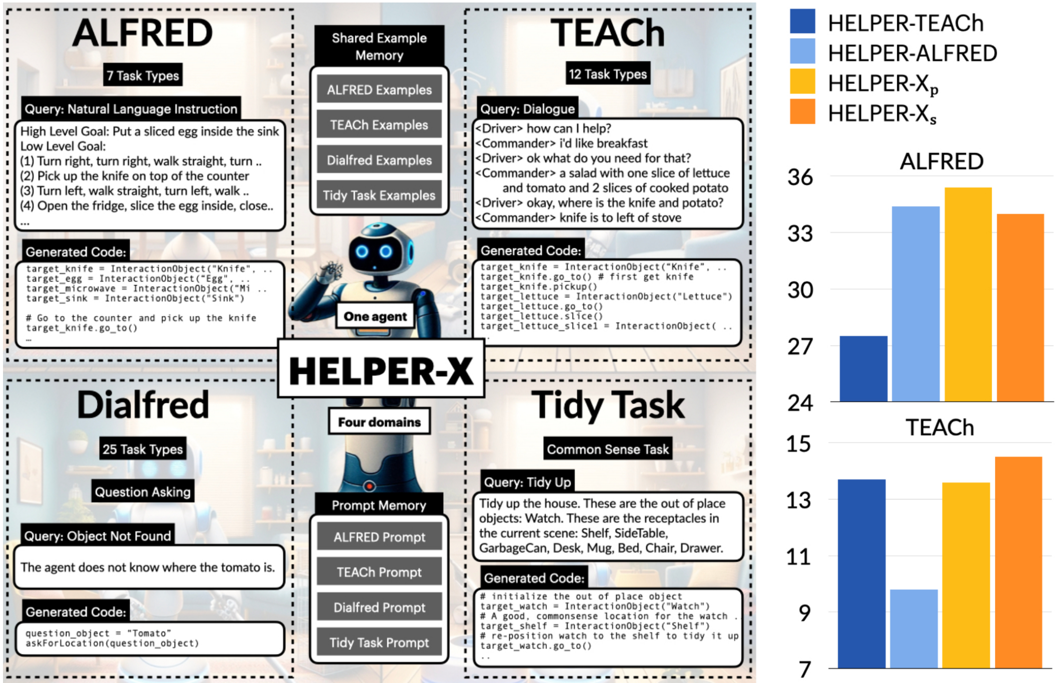

Figure 1: **(left)** HELPER-X can perform dialogue-based task completion in TEACh (Padmakumar, 2021), follow instructions from natural language in ALFRED (Shridhar, 2020), engage in instruction following with active question asking in DialFRED (Gao, 2022), and organize rooms using spatial commonsense reasoning in the Tidy Task (Sarch, 2022) with a single model. **(right)** Prompted GPT-4 cannot generalize from an action-from-dialogue domain (TEACh) to action-from-instruction domain (ALFRED). TEACh-tailored HELPER Sarch (2023), with corresponding prompt templates and example memory, shows a 6.9% decrease in success rate when tested on ALFRED, a related domain that shares the same action space and environments but differs in language inputs and task types. HELPER-X maintains strong performance across both domains with a single model.

carefully designed, either by manual search or an automated procedure (Jiang, 2020; Shin, 2020). Consequently, the challenge with adapting LLMs to a particular task becomes providing appropriate prompts for the model to understand the task constraints and generate accurate outputs. For embodied agent task planning, LLMs are commonly given a prompt that includes a description of the task to perform, an API that defines the action space with function documentation, environment and task instruction inputs, and a set of in-context examples for expected inputs and planning outputs for the task (Liang, 2022; Singh, 2023).

When the amount of in-context examples and task descriptions necessary to cover the task constraints increases, inference costs significantly rise due to additional attention operations. To handle computational challenges and LLM context length, a growing body of research explores the concept of "memory-augmented prompting" – a method that involves retrieving a set of pertinent in-context examples to append to the prompt, thereby broadening their applicability (Perez, 2021; Schick, 2020; Gao, 2020; Liu, 2022c; Song, 2023; Sarch, 2023; Lewis, 2020; Mao, 2021). For example, HELPER in Sarch (2023), retrieves a set of language-program examples based on the user's input instruction and adds them to the prompt to provide contextualized examples for GPT-4 task planning.

Despite GPT-4's robust generalization, memory-augmented prompting tailored for one domain does not guarantee high performance in a similar yet distinct domain. Applying HELPER, prompted with TEACh-specific examples and descriptions, to ALFRED – a related domain that shares the same action space and environments but differs in language inputs and task types – results in a notable 6.9% decrease in accuracy compared to a HELPER model with a specialized prompt and customized example memory specifically for ALFRED, and vice versa when doing the same on TEACh (3.2%

decrease), as shown in Figure 1 (right). To further address the question of whether a *single agent* with an expanded prompt and example memory can effectively operate across multiple benchmarks,

we introduce HELPER-X, an embodied agent capable of executing plans from dialogue, following natural language instructions, asking questions actively, and organizing rooms, using spatial commonsense reasoning. HELPER-X extends the capabilities of HELPER, by expanding its memory with a wider array of examples and prompts, and by integrating a additional APIs for asking questions. Specifically, we introduce two HELPER-X variants: HELPER-X$_P$, which retrieves domain-specific prompt templates and related in-context examples for the LLM, and HELPER-X$_S$, which retrieves in-context examples from a shared memory under a single prompt template.

We evaluate HELPER-X across four domains that include dialogue-based task completion on TEACh (Padmakumar, 2021), following instructions from natural language on ALFRED (Shridhar, 2020), engaging in instruction following with active question asking on DialFRED (Gao, 2022), and organizing rooms using spatial commonsense reasoning in the Tidy Task (Sarch, 2022). HELPER-X demonstrates state-of-the-art performance in the few-shot example domain, that is, without in-domain training. Extending the language-program memory does not cause interference and does not hinder performance. In fact, HELPER-X matches, and sometimes even exceeds, the performance of agents prompted with a single-domain in mind.

HELPER inherits the key advantages of prompting and memory-augmentation: 1) *generality*, as it unifies many embodied agent tasks into one system; 2) *training-free*, as it does not require re-training (or fine-tuning) a new model for every new task; 3) *interpretability*, as the planning is explicit code and the retrieval system identifies the examples and prompts used for planning; 4) *adaptability*, by being able to continually add to the memory and APIs to incorporate new inputs, skills, and tasks; 5) *flexibility*, as the architecture is modular, improvements in any of the used vision and language modules will result in direct improvement in HELPER-X's performance.

2 Related Work

2.1 Memory-Augmented Prompting of Large Language Models

Recently, external memories have been instrumental in scaling language models Borgeaud (2021); Khandelwal (2019), overcoming the constraints of limited context windows in parametric transformers Wu (2022). They also facilitate knowledge storage in various forms such as entity mentions de Jong (2021), knowledge graphs Das (2022), and question-answer pairs Chen (2022). Retrieval-augmented generation (RAG) (Lewis, 2020; Mao, 2021) has been shown to significantly improve response quality in large language models (LLMs) by integrating external knowledge sources with the model's internal representations. In agent-based domains, memory-augmented prompting has enhanced task planning in embodied instructional contexts (Song, 2023; Sarch, 2023) and open-world gaming (Wang, 2023b;a; Majumder, 2023). Our model, HELPER-X, employs memory-augmented prompting across four benchmarks, demonstrating that memory expansion across related domains can maintain performance.

2.2 Instructable Embodied Agents that Interact with their Environments

Numerous benchmarks assess embodied vision-and-language tasks, with significant advancements in learning-based embodied AI agents across tasks like scene rearrangement Gan (2022); Weihs (2021); Batra (2020); Sarch (2022); Trabucco (2022), object-goal navigation Anderson (2018); Yang (2019); Wortsman (2019); Chaplot (2020); Gupta (2017); Chang (2020); Gervet (2023); Chang (2023), point-goal navigation and exploration Anderson (2018); Savva (2019); Wijmans (2020); Ramakrishnan (2020); Gupta (2017); Chen (2019); Chaplot (2019); Kumar (2021), embodied question answering Gordon (2018); Das (2018); Zhu (2023); Datta (2022); Das (2020); Gao (2022), instructional and image navigation (Ku, 2020; Krantz, 2023), audio-visual navigation (Chen, 2020), interactive dialogue and natural language instruction following (Yenamandra, 2023; Shridhar, 2020; Padmakumar, 2021; Gao, 2023), and embodied commonsense reasoning (Kant, 2022; Sarch, 2022; Wu, 2023). Interactive instruction benchmarks (e.g., ALFRED (Shridhar, 2020) and TEACh (Padmakumar, 2021)) require agents to follow natural language directives and dialogue, identifying objects in scenes via interaction masks. Variants like DialFRED (Gao, 2022) allow agent inquiries about objects and locations. Benchmarks such as TIDEE (Sarch, 2022) and HouseKeep (Kant, 2022) test agents' ability to tidy rooms using commonsense, without explicit object placement directives. Unlike most methods confined to a single domain, our work focuses on creating a multi-domain agent adept in dialogue-based task planning, natural language instruction following, asking questions for disambiguation of instructions, and tidying up scenes. Our method shows competitive performance across the four domains with a few task-specific demonstrations and without domain-specific weights, beyond the single image object detector.

Interactive vision-language embodied agent methods train distinct agents for each language-defined task, using large datasets from expert demonstrations (Min, 2021; Inoue, 2022; Zhang, 2022; Kim, 2023; Pashevich, 2021). Some approaches use these demonstrations for end-to-end network training to directly predict actions from observations (Pashevich, 2021; Gao, 2022; Zhang, 2021). Others employ modular methods, training planners to generate subgoals handled by specialized perception, manipulation, and obstacle avoidance modules (Min, 2021; Inoue, 2022; Blukis, 2022; Zheng, 2022; Kim, 2023; Bhambri, 2023; Liu, 2022a; Murray, 2022; Liu, 2023). However, these methods often over-specialize to specific datasets and tasks, limited by the training domain's language and task structure. In contrast, our method performs competitively across multiple benchmarks with minimal task-specific demonstrations and without needing domain-specific networks.

3 Our Method: HELPER-X

We extend HELPER (Sarch, 2023) to work across four domains. We propose two versions to extend the memory-augmented prompting of LLMs in HELPER: 1) HELPER-X$_P$ that retrieves from a memory of domain-tailored prompt templates and associated domain-specific examples (Section 3.2.1), and 2) HELPER-X$_S$ that expands the memory of HELPER into a shared memory of in-context examples across domains combined with a domain-agnostic prompt template (Section 3.2.2). Additionally, we extend the capabilities of HELPER for question asking, by appending a question API with functions defining possible questions and their arguments to the LLM prompt (Section 3.2.3). We use HELPER (Sarch, 2023) for execution of the generated program using standard perception modules.

3.1 Background: HELPER

We build on HELPER (Sarch, 2023) for memory-augmented prompting of LLMs and program execution. Here, we give an account of HELPER to make the paper self-contained.

HELPER prompts an LLM, namely GPT-4 (gpt, 2023), to generate plans as Python programs. It assumes that the agent has access to a set of action skills S (e.g., `go_to(X)`, `pickup(X)`, etc.). HELPER adds these skills to the prompt in the form of a Python API. The LLM is instructed only to call these pre-defined skill functions in its generated programs. HELPER considers a key-value memory of language inputs and successful program pairs. It retrieves a set of in-context examples relevant to the current input language to add to the prompt to assist program generation. Each key is encoded into a language embedding. The top-k language-program pairs are retrieved based on their euclidean distance to the encoding of the language input I encoding.The HELPER prompt also contains a role description ("You are a helpful assistant with expertise in..."), a task description ("Your task is to ...") and guidelines to follow for program generation ("You should only use functions in the API..."), that are commonly tailored to the domain-of-interest.

HELPER (Sarch, 2023), using RGB input, estimates depth maps, object masks, and agent egomotion at each timestep. This facilitates the creation and upkeep of a 3D occupancy map and an object memory database, essential for obstacle navigation and object tracking. Object detection in each frame leads to instance aggregation based on 3D centroid proximity, with each instance characterized by dynamic state attributes (e.g., cooked, sliced, dirty). When an action fails, a Vision-Language Model (CLIP (Radford, 2021)) provides feedback, prompting the LLM to re-plan. For objects not present in the map, the LLM suggests areas for HELPER to search (e.g., “near the sink").

3.2 Unified Memory-Augmented Prompting

We explore two ways to expand HELPER to work across four domains, either through prompt retrieval (Section 3.2.1) or through a shared example memory (Section 3.2.2).

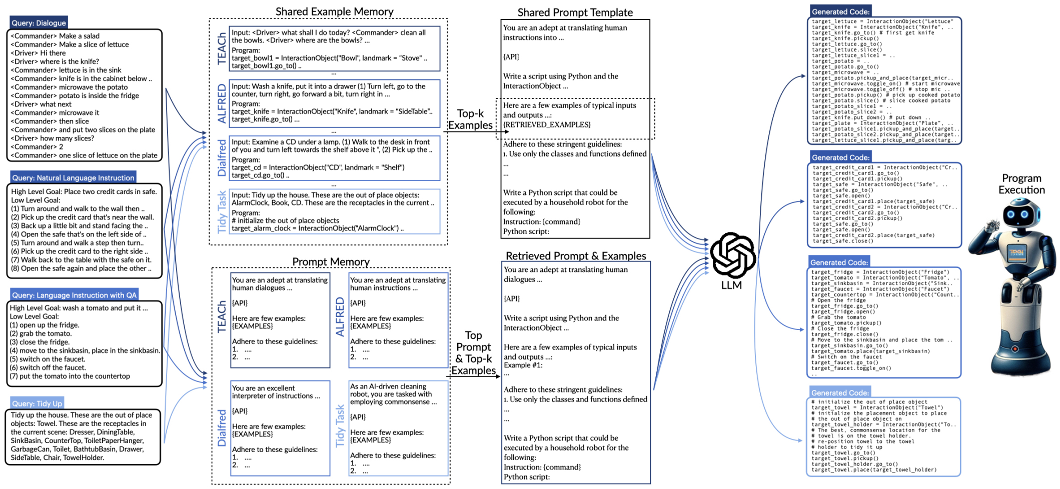

Figure 2: Illustration of the shared example memory (HELPER-X$_S$; top) and the prompt retrieval (HELPER-X$_P$; bottom). The memory is shared across domains in both versions, allowing language and task inputs from any of the domains.

3.2.1 Prompt Retrieval

Given an input language instruction I, the prompt retrieval agent HELPER-X$_P$ retrieves a specialized prompt template P and an associated set of specialized examples E. Each specialized prompt template contains role descriptions, task instructions and guidelines tailored to each domain, namely, dialogue-based task completion (based on TEACh (Padmakumar, 2021)), instruction following from natural language (based on ALFRED (Shridhar, 2020)), instruction following with active question asking (based on DialFRED (Gao, 2022)), or tidying up rooms (based on Tidy Task (Sarch, 2022)). For retrieval, a query is generated from the input instruction by encoding the instruction into an embedding vector using a pre-trained language model (Greene, 2022). The query retrieves the closest key from memory, where each key represents the language encodings of each prompt template and example text , as shown in Figure 2. The top-k in-context examples are further retrieved from the specialized set of examples associated with the retrieved prompt and added to the retrieved prompt template, and the resulting prompt is used for LLM's program generation.

3.2.2 Shared Example Memory

Given an input language instruction I, the shared example memory agent HELPER-X$_S$ retrieves a set of in-context examples from a shared memory that includes in-context examples from all domains considered. These examples are added to a domain-agnostic prompt template that does not have a specialized role description, task instructions or guidelines for any single domain. A query is generated from the input instruction by encoding it into an embedding vector using a pre-trained language model (Greene, 2022). The keys represent encodings of each in-context example language in the shared memory. The query embedding retrieves the top-k nearest neighbors keys and their values. These are added to the prompt as relevant in-context examples for LLM program generation, as shown in Figure 2.

3.2.3 Question Asking API

A natural limitation of asking questions in a simulator is that only certain types of questions can be understood and answered. We constrained the set of possible questions asked by the agent by defining an API of available questions in the DialFRED (Gao, 2022) benchmark and their arguments that HELPER-X can call on to gather more information. These include questions in three categories—Location, Appearance, and Direction—pertaining to the agent's next interaction object. Importantly, this API can be continuously expanded by adding an additional function to the question-asking API.

Implementation details. We follow the network implementation of HELPER (Sarch, 2023). We use `GPT-4-0613` (gpt, 2023) for text generation and `text-embedding-ada-002` (Greene, 2022) for text embeddings. We use the SOLQ object detector (Dong, 2021) and ZoeDepth network (Bhat, 2023) for depth estimation from RGB input. We use $k = 3$ for example retrieval.

4 EXPERIMENTS

We test HELPER-X in the following benchmarks: 1. Inferring and executing action plans from dialogue (TEACh (Padmakumar, 2021)), 2. Inferring and executing action plans from instructions (ALFRED (Shridhar, 2020)), 3. Active question asking for seeking help during instruction execution (DialFRED (Gao, 2022)), and 4. Tidying up rooms (Tidy Task (Sarch, 2022)).

4.1 INFERRING AND EXECUTING ACTION PLANS FROM DIALOGUE

Inferring and executing task plans from dialogue involves understanding dialogue segments and executing related instructions using the provided information in the dialogue. This task evaluates the agent's ability in understanding noisy and free-form conversations between two humans discussing about a household task.

Dataset We use the TEACh benchmark (Padmakumar, 2021), which consists of over 3,000 dialogues focused on household tasks in the AI2-THOR environment Kolve (2017). We use the Trajectory from Dialogue (TfD) variant, where an agent, given a dialogue segment, must infer action sequences to fulfill tasks like making a coffee or preparing a salad. The training dataset contains 1482 expert demonstrations with associated dialogues. The evaluation includes 181 'seen' and 612 'unseen' episodes, with 'seen' having different object placements and states than in training, and 'unseen' also having different object instances and room environments. The agent receives an egocentric image at each step and selects actions to execute, such as `pickup(X)`, `turn_left()`, etc.

Baselines We consider two kinds of baselines: **1.** Methods that supervise low-level or high-level action prediction from language and visual input using expert demonstrations in the training set (1482 in number) (Pashevich, 2021; Zheng, 2022; Min, 2021; Zhang, 2022), and **2.** Methods that use a small amount of expert demonstrations for prompting pretrained LLM (Sarch, 2023). Specifically, HELPER and HELPER-X use 11 domain-specific examples. We additionally include a comparison to HELPER-ALF, which uses specialized prompts and examples for ALFRED.

Evaluation Metrics We follow the TEACh evaluation metrics. **Task success rate (SR)** is a binary metric of whether all subtasks were successfully completed. **Goal condition success rate (GC)** quantifies the proportion of achieved goal conditions across all sessions. Both SR and GC have path-length weighted variants weighted by (path length of the expert trajectory) / (path length taken by the agent).

Results are reported in Table 1. **On validation unseen, HELPER-X$_S$ and HELPER-X$_P$ demonstrate performance on-par with HELPER, with HELPER-X$_S$ even slightly outperforming HELPER,** despite HELPER-X being shared across four domains. HELPER-X also outperforms the best supervised baselines trained in-domain with many demonstrations. **On validation seen, both HELPER-X variants outperform HELPER, with the best model HELPER-X$_P$ outperforming HELPER by 2.7% in success rate.**.

4.2 FOLLOWING NATURAL LANGUAGE INSTRUCTIONS

Natural language instruction following evaluates the agent's ability to carry out high-level instructions ("Rinse off a mug and place it in the coffee maker") and low-level ones ("Walk to the coffee maker on the right") provided by a human user. Importantly, the language and tasks in this evaluation differ from the ones in the TEACh benchmark (Section 4.1).

Dataset Te ALFRED (Shridhar, 2020) is a vision-and-language navigation benchmark designed for embodied agents to execute tasks in domestic settings from RGB sensory input. It includes seven task types across 207 environments, that involve 115 object types in 4,703 task instances, varying from simple object relocation to placing a heated item in a receptacle. The dataset includes detailed

Table 1: **Trajectory from Dialogue (TfD) evaluation on the TEACh validation set.** Trajectory length weighted metrics are included in (parentheses). FS = few shot. Sup. = supervised. G = generalist; shared across benchmarks. GC = goal-condition success.

		Unseen		Seen	
		Success	GC	Success	GC
Sup.	E.T. (Pashevich, 2021)	0.5 (0.1)	0.4 (0.6)	1.0 (0.2)	1.4 (4.8)
	JARVIS (Zheng, 2022)	1.8 (0.3)	3.1 (1.6)	1.7 (0.2)	5.4 (4.5)
	FILM (Min, 2022)	2.9 (1.0)	6.1 (2.5)	5.5 (2.6)	5.8 (11.6)
	DANLI (Zhang, 2022)	8.0 (3.2)	6.8 (6.6)	5.0 (1.9)	10.5 (10.3)
	ECLAIR (Kim, 2023)	13.2	–	–	–
FS	HELPER-ALF	10.5 (1.8)	10.3 (4.5)	13.3 (2.8)	14.2 (7.5)
	HELPER (Sarch, 2023)	13.7 (1.6)	**14.2 (4.6)**	12.2 (1.8)	18.6 (9.3)
G+FS	HELPER-X$_P$	13.6 (2.0)	13.6 (5.6)	**14.9 (3.6)**	**20.3 (11.0)**
	HELPER-X$_S$	**14.5 (2.1)**	14.0 (5.4)	14.4 (3.5)	19.9 (11.0)

human-authored instructions and high-level goals, based on 21,023 expert demonstrations. It also comprises 820 'seen' and 821 'unseen' validation episodes. Agents receive egocentric RGB images at each step and select actions from a predefined set to progress, such as `pickup(X)`, `turn_left()`, etc.

Baselines Again, we consider two sets of baselines: those that supervise low-level or high-level action prediction using the expert demonstrations in the training set (Pashevich, 2021; Zheng, 2022; Min, 2021; Zhang, 2022; 2021; Song, 2022a; Blukis, 2022; Bhambri, 2023; Liu, 2022a; Murray, 2022; Inoue, 2022), and those that use a small amount of demonstrations ($<= 100$) (few-shot) (Sarch, 2023; Song, 2023; Brohan, 2023; Liu, 2023). The SayCan (Brohan, 2023), FILM (Min, 2021) (FS), and HLSM (Blukis, 2022) (FS) few shot baselines are adapted for the few-shot ALFRED setting by the authors of Song (2023), where the planning modules in FILM and HLSM are re-trained using only 100 demonstrations. We adapt HELPER (Sarch, 2023) as a baseline with specialized prompts and examples for ALFRED, as well as a comparison to HELPER-TEACh, which uses specialized prompts and examples for TEACh. HELPER and our HELPER-X model each use 7 domain-specific examples in their memory.

Evaluation Metrics We follow the ALFRED evaluation metrics: **1. Task success rate (SR)** and **2. Goal condition success rate (GC)**. These are defined the same as in TEACh (Section 4.1).

Results are reported in Table 2. Our conclusions are similar to Section 4.1. On validation unseen, HELPER-X$_S$ and HELPER-X$_P$ demonstrate performance on-par with HELPER, with HELPER-X$_P$ marginally outperforming HELPER by 1.0%, despite. HELPER-X is also competitive with the best supervised baselines, despite only requiring a few in-domain demonstrations. On validation seen, we observe both HELPER-X models marginally outperforming HELPER. We additionally show that using HELPER-TEACh which has prompts and examples for a different domain (TEACh) causes a significant 6.9% drop in performance.

4.3 Instruction Following with Asking Questions

Question asking instruction following allows the agent to choose to ask questions to an oracle to gain additional information to help it complete a task defined by an initial natural language instruction.

Dataset The DialFRED benchmark (Gao, 2022) enables an agent to query users while executing language instructions, utilizing user responses for task improvement. It features a human-annotated dataset with 53K relevant questions and answers, plus an oracle for responding to agent queries. Agents can ask questions in three categories—Location, Appearance, and Direction—pertaining to their next interaction object. The dataset covers 25 task types across 207 environments, 115 object types, and includes 'seen' and 'unseen' episodes. The agent receives egocentric RGB images at each step and selects actions from a set, like `pickup(X)`, `turn_left()`, etc. This benchmark's instructions and tasks are distinct from TEACh and partially overlap with ALFRED, with significant modifications and 18 new task types.

Table 2: **Evaluation on the ALFRED validation unseen set.** Trajectory length weighted metrics are included in (parentheses). FS = few shot. Sup. = supervised. G = generalist; shared across benchmarks.

		Unseen		Seen	
		Success	GC	Success	GC
Sup.	E.T. (Pashevich, 2021)	7.3 (3.3)	20.9 (11.3)	46.6 (32.3)	52.9 (42.2)
	HiTUT (Zhang, 2021)	12.4 (6.9)	23.7 (12.0)	25.2 (12.2)	34.9 (18.5)
	M-TRACK(Song, 2022a)	17.29	28.98	26.70	33.21
	HLSM (Blukis, 2022)	18.3	31.2	29.6	38.7
	FILM (Min, 2021)	20.1	32.5	24.6	37.2
	MCR-Agent (Bhambri, 2023)	20.1 (10.8)	–	34.4 (23.0)	–
	LEBP (Liu, 2022a)	22.36	29.58	27.63	35.76
	LGS-RPA (Murray, 2022)	33.18	44.68	43.86	52.51
	EPA (Liu, 2023)	40.11	44.14	45.78	51.03
	Prompter (Inoue, 2022)	53.3 (19.6)	63.0 (21.7)	–	–
FS	HLSM (Blukis, 2022) (FS)	0.00	1.86	0.1	2.8
	FILM (Min, 2021) (FS)	0.00	9.65	0.0	0.0
	SayCan (Brohan, 2023)	9.9	22.5	12.3	24.5
	LLM-Planner (Song, 2023)	15.4	23.4	16.5	30.1
	HELPER-TEACh	27.5 (5.9)	44.3 (10.1)	24.5 (6.3)	38.2 (11.2)
	HELPER (Sarch, 2023)	34.4 (7.6)	51.5 (11.9)	27.6 (7.4)	42.0 (12.7)
G+FS	HELPER-X$_P$	**35.4 (7.9)**	**52.9 (12.3)**	**28.2 (7.5)**	**42.5 (12.9)**
	HELPER-X$_S$	34.0 (7.5)	51.1 (11.9)	28.0 (7.4)	42.1 (12.8)

Questioning Implementation To ask questions in the DialFRED task, we add to the question asking API functions to query the oracle in DialFRED (see Section 3.2.3). HELPER-X asks questions when it does not know the location of an object required for the task at hand. Unlike ALFRED, success in DialFRED requires interacting with a specific instance of an object class. To account for this, HELPER-X also asks questions to help disambiguate when it has seen multiple instances of the same object.

Baselines We compare with the baselines in the DialFRED paper (Gao, 2022), which includes a sequence-to-sequence architecture for choosing to ask a question, trained with reinforcement learning, and the Episodic Transformer architecture (Pashevich, 2021), trained with behavioral cloning. We adapt HELPER (Sarch, 2023) as a baseline with specialized prompts and examples for DialFRED, as well as our question asking API. We consider a few shot setting with each few shot model receiving 7 domain-specific examples.

Evaluation Metrics We follow the conventions of the DialFRED benchmark. We use the **Task success rate (SR)** metric. This is defined the same as TEACh (Section 4.1).

Results are reported in Table 3. On validation unseen, we observe HELPER-X$_S$ marginally outperforming HELPER by 0.38 points in success rate, despite HELPER-X being shared across all domains. While HELPER-X is outperformed by the best supervised baselines, HELPER-X only requires a few in-domain demonstrations compared to the thousands of language-action demonstrations and RL interactions needed to train the baseline models. Most importantly, we see the addition of question-asking in HELPER-X improves success rate by 2.48 points, highlighting its efficiency in question selection and response utilization.

4.4 Tidying Up using Spatial Commonsense Reasoning

Tidying up involves figuring out where to place items without explicit instructions, relying on spatial commonsense to infer a proper location for an object. This task tests the agent's ability to use commonsense reasoning regarding contextual, object-object, and object-room spatial relations.

Dataset We evaluate on the Tidy Task (Sarch, 2022) benchmark, where the agent is spawned in a disorganized room, and must reposition objects to bring them to an organized tidy state. The dataset consists of 8000 training, 200 validation, and 100 testing messy configurations in 120 distinct

Table 3: **Evaluation on the DialFRED validation unseen set.** FS = few shot. Sup. = supervised. G = generalist; shared across benchmarks.

		Success
Sup.	Instructions Only (Gao, 2022)	18.3
	All QAs (Gao, 2022)	32.0
	RL Anytime (Gao, 2022)	33.6
FS	HELPER (Sarch, 2023)	19.62
G+FS	HELPER-X$_P$	18.96
	without QA	16.48
	HELPER-X$_S$	**19.99**

Table 4: **Evaluation on the Tidy Task test set.** Trajectory length weighted metrics are included in (parentheses). FS = few shot. S = supervised. G = generalist; shared across benchmarks. CM = Correctly Moved. IM = Incorrectly Moved.

		CM ↑	IM ↓	Energy% ↓	Steps
S	TIDEE (Sarch, 2022)	2.7	0.3	64.9	437.6
FS	Random Receptacle	2.0	0.3	95.5	329.3
	HELPER (Sarch, 2023)	2.1	**0.2**	83.9	348.3
G+FS	HELPER-X$_P$	2.1	0.3	86.9	368.2
	HELPER-X$_S$	**2.2**	**0.2**	**83.4**	333.9

scenes of bedrooms, living rooms, kitchens and bathrooms. At each time step, the agent obtains an egocentric RGB and depth image and must choose an action from a specified set to transition to the next step, such as `pickup(X)`, `turn_left()`, etc. In this setup, the models are prompted to tidy up the room, given a set of objects that are out of place obtained using the visual detector from TIDEE (Sarch, 2022).

Baselines We compare against TIDEE (Sarch, 2022), which includes a graph neural network encoding common object arrangements. This is supervised in the training set of the Tidy Task to predict where a target object should be re-positioned to in the current scene. We adapt HELPER (Sarch, 2023) as a baseline with specialized prompts and examples for the Tidy Task. We additionally include a random receptacle baseline which chooses random receptacle placement locations for the out of place objects. We consider a few shot setting with each few shot model receiving 3 domain-specific examples.

Evaluation Metrics We use the following evaluation metrics for the Tidy Task: **Correctly Moved (CM)** Average number of correctly moved objects that are out of place in the scene, and moved by the agent. Higher is better. **Incorrectly Moved (IM)** Average number of incorrectly moved objects that are not out of place, but were moved by the agent. Lower is better. **Energy** The "cleaniness" energy, where lower energy represents a higher likelihood of the room object configuration aligning with the configurations in the organized AI2THOR rooms. Following ProcThor (Deitke, 2022), for each receptacle object, the probability that each object type appears on its surface is computed across the AI2THOR scenes. See the Appendix for more details.

Results are in Table 4. On the Tidy Task, HELPER-X$_S$ and HELPER-X$_P$ demonstrates performance on-par with HELPER. HELPER-X does significantly better than if object locations are randomly placed (Random Receptacle). We find that the supervised baseline, TIDEE, outperforms HELPER-X, especially in the Energy metric, revealing that in-domain training on this benchmark is helpful for learning the common object configurations within the AI2THOR environments. However, we find that HELPER-X accomplishes the task in significantly fewer steps compared to TIDEE.

5 CONCLUSION

We introduce HELPER-X, an embodied agent that executes tasks from dialogue or language instructions, ask questions, and tides up rooms. HELPER-X has two variants, HELPER-X$_P$ and HELPER-X$_S$, enhancing HELPER's memory capabilities. HELPER-X$_P$ retrieves domain-specific templates and examples for large language models, while HELPER-X$_S$ retrieves only examples for a domain-agnostic prompt template through a shared memory. Evaluation of HELPER-X in four domains: TEACh, ALFRED, DialFRED, and the Tidy Task, yields state-of-the-art performance in the few-shot example setting. Memory and API expansions we considered maintained or improved performance for the LLM, highlighting the effectiveness of memory-enhanced LLMs in building versatile, instructable agents.

6 ACKNOWLEDGEMENTS

This material is based upon work supported by National Science Foundation grants GRF DGE1745016 & DGE2140739 (GS), a DARPA Young Faculty Award, a NSF CAREER award, an AFOSR Young Investigator Award, AFOSR FA9550-20-1-0423 award, a DARPA Machine Common Sense, and an ONR award N000142312415. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Army, the National Science Foundation, or the United States Air Force.

This research project has benefitted from the Microsoft Accelerate Foundation Models Research (AFMR) grant program through which leading foundation models hosted by Microsoft Azure along with access to Azure credits were provided to conduct the research.

The authors thank William W. Cohen for discussions and useful feedback over the course of this project.

REFERENCES

Openai. gpt-4 technical report. *arXiv preprint arxiv:2303.08774*, 2023.

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018.

Dhruv Batra, Angel Xuan Chang, S. Chernova, Andrew J. Davison, Jun Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, Manolis Savva, and Hao Su. Rearrangement: A challenge for embodied ai. *ArXiv*, abs/2011.01975, 2020.

James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions, 2023.

Suvaansh Bhambri, Byeonghwi Kim, and Jonghyun Choi. Multi-level compositional reasoning for interactive instruction following. *Interaction*, 3:4, 2023.

Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. Zoedepth: Zero-shot transfer by combining relative and metric depth. *arXiv preprint arXiv:2302.12288*, 2023.

Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *Conference on Robot Learning*, pp. 706–717. PMLR, 2022.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. *CoRR*, abs/2112.04426, 2021. URL `https://arxiv.org/abs/2112.04426`.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pp. 287–318. PMLR, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Matthew Chang, Arjun Gupta, and Saurabh Gupta. Semantic visual navigation by watching youtube videos. *Advances in Neural Information Processing Systems*, 33:4283–4294, 2020.

Matthew Chang, Theophile Gervet, Mukul Khanna, Sriram Yenamandra, Dhruv Shah, So Yeon Min, Kavit Shah, Chris Paxton, Saurabh Gupta, Dhruv Batra, Roozbeh Mottaghi, Jitendra Malik, and Devendra Singh Chaplot. Goat: Go to any thing, 2023.

Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. In *International Conference on Learning Representations*, 2019.

Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems*, 33, 2020.

Changan Chen, Unnat Jain, Carl Schissler, Sebastia Vicenc Amengual Gari, Ziad Al-Halah, Vamsi Krishna Ithapu, Philip Robinson, and Kristen Grauman. Soundspaces: Audio-visual navigation in 3d environments. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, pp. 17–36. Springer, 2020.

Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/pdf?id=SyMWn05F7.

Wenhu Chen, Pat Verga, Michiel de Jong, John Wieting, and William Cohen. Augmenting pre-trained language models with qa-memory for open-domain question answering, 2022. URL https://arxiv.org/abs/2204.04581.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–10, 2018.

Abhishek Das, Federico Carnevale, Hamza Merzic, Laura Rimell, Rosalia Schneider, Josh Abramson, Alden Hung, Arun Ahuja, Stephen Clark, Gregory Wayne, et al. Probing emergent semantics in predictive agents via question answering. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 2376–2391, 2020.

Rajarshi Das, Ameya Godbole, Ankita Naik, Elliot Tower, Robin Jia, Manzil Zaheer, Hannaneh Hajishirzi, and Andrew McCallum. Knowledge base question answering by case-based reasoning over subgraphs, 2022. URL https://arxiv.org/abs/2202.10610.

Samyak Datta, Sameer Dharur, Vincent Cartillier, Ruta Desai, Mukul Khanna, Dhruv Batra, and Devi Parikh. Episodic memory question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19119–19128, 2022.

Michiel de Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Fei Sha, and William Cohen. Mention memory: incorporating textual knowledge into transformers through entity mention attention. *CoRR*, abs/2110.06176, 2021. URL https://arxiv.org/abs/2110.06176.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994, 2022.

Bin Dong, Fangao Zeng, Tiancai Wang, Xiangyu Zhang, and Yichen Wei. Solq: Segmenting objects by learning queries. *Advances in Neural Information Processing Systems*, 34:21898–21909, 2021.

Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel L.K. Yamins, James J. DiCarlo, Josh McDermott, Antonio Torralba, and Joshua B. Tenenbaum. The threedworld transport challenge: A visually guided task-and-motion planning benchmark towards physically realistic embodied ai. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 8847–8854, 2022. doi: 10.1109/ICRA46639.2022.9812329.

Qiaozi Gao, Govind Thattai, Xiaofeng Gao, Suhaila Shakiah, Shreyas Pansare, Vasu Sharma, Gaurav Sukhatme, Hangjie Shi, Bofei Yang, Desheng Zheng, et al. Alexa arena: A user-centric interactive platform for embodied ai. *arXiv preprint arXiv:2303.01586*, 2023.

Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. *arXiv preprint arXiv:2012.15723*, 2020.

Xiaofeng Gao, Qiaozi Gao, Ran Gong, Kaixiang Lin, Govind Thattai, and Gaurav S Sukhatme. DialFRED: Dialogue-enabled agents for embodied instruction following. *IEEE Robotics and Automation Letters*, 7(4):10049–10056, 2022.

Theophile Gervet, Soumith Chintala, Dhruv Batra, Jitendra Malik, and Devendra Singh Chaplot. Navigating to objects in the real world. *Science Robotics*, 8(79):eadf6991, 2023.

Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. Iqa: Visual question answering in interactive environments. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4089–4098, 2018.

Ryan Greene, Ted Sanders, Lilian Weng, and Arvind Neelakantan. Openai. new and improved embedding model. 2022.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

SU Hongjin, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. Selective annotation makes language models better few-shot learners. In *The Eleventh International Conference on Learning Representations*, 2022.

Yuki Inoue and Hiroki Ohashi. Prompter: Utilizing large language model prompting for a data efficient embodied instruction following. *arXiv preprint arXiv:2211.03267*, 2022.

Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

Yash Kant, Arun Ramachandran, Sriram Yenamandra, Igor Gilitschenski, Dhruv Batra, Andrew Szot, and Harsh Agrawal. Housekeep: Tidying virtual households using commonsense reasoning. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIX*, pp. 355–373. Springer, 2022.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models, 2019. URL `https://arxiv.org/abs/1911.00172`.

Byeonghwi Kim, Jinyeon Kim, Yuyeong Kim, Cheolhong Min, and Jonghyun Choi. Context-aware planning and environment-aware memory for instruction following embodied agents. In *ICCV*, 2023.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017.

Jacob Krantz, Theophile Gervet, Karmesh Yadav, Austin Wang, Chris Paxton, Roozbeh Mottaghi, Dhruv Batra, Jitendra Malik, Stefan Lee, and Devendra Singh Chaplot. Navigating to objects specified by images. *arXiv preprint arXiv:2304.01192*, 2023.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4392–4412, 2020.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. 2021.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

Haoyu Liu, Yang Liu, Hongkai He, and Hangfang Yang. Lebp–language expectation & binding policy: A two-stream framework for embodied vision-and-language interaction task learning agents. *arXiv preprint arXiv:2203.04637*, 2022a.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *DeeLIO 2022*, pp. 100, 2022b.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, William B Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pp. 100–114, 2022c.

Xiaotian Liu, Hector Palacios, and Christian Muise. Egocentric planning for scalable embodied task achievement. *arXiv preprint arXiv:2306.01295*, 2023.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Burch Callison-Burch, and Peter Clark. Clin: A continually learning language agent for rapid task adaptation and generalization. *arXiv*, 2023.

Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. Generation-augmented retrieval for open-domain question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4089–4100, 2021.

So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. Film: Following instructions in language with modular methods, 2021.

So Yeon Min, Hao Zhu, Ruslan Salakhutdinov, and Yonatan Bisk. Don't copy the teacher: Data and model challenges in embodied dialogue. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 9361–9368, 2022.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3470–3487, 2022.

Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *arXiv preprint arXiv:2305.15021*, 2023.

Michael Murray and Maya Cakmak. Following natural language instructions for household tasks with landmark guided search and reinforced pose adjustment. *IEEE Robotics and Automation Letters*, 7(3):6870–6877, 2022.

OpenAI. Gpt-4 technical report, 2023.

Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat, 2021.

Alexander Pashevich, Cordelia Schmid, and Chen Sun. Episodic transformer for vision-and-language navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15942–15952, 2021.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. *Advances in neural information processing systems*, 34:11054–11070, 2021.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Santhosh K Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation. In *European Conference on Computer Vision*, pp. 400–418. Springer, 2020.

Gabriel Sarch, Zhaoyuan Fang, Adam W Harley, Paul Schydlo, Michael J Tarr, Saurabh Gupta, and Katerina Fragkiadaki. Tidee: Tidying up novel rooms using visuo-semantic commonsense priors. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIX*, pp. 480–496. Springer, 2022.

Gabriel Sarch, Yue Wu, Michael Tarr, and Katerina Fragkiadaki. Open-ended instructable embodied agents with memory-augmented large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9339–9347, 2019.

Timo Schick and Hinrich Schütze. It's not just size that matters: Small language models are also few-shot learners. *arXiv preprint arXiv:2009.07118*, 2020.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4222–4235, 2020.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023.

Chan Hee Song, Jihyung Kil, Tai-Yu Pan, Brian M Sadler, Wei-Lun Chao, and Yu Su. One step at a time: Long-horizon vision-and-language navigation with milestones. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15482–15491, 2022a.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models, 2023.

Haoyu Song, Li Dong, Weinan Zhang, Ting Liu, and Furu Wei. Clip models are few-shot learners: Empirical studies on vqa and visual entailment. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6088–6100, 2022b.

Brandon Trabucco, Gunnar A Sigurdsson, Robinson Piramuthu, Gaurav S Sukhatme, and Ruslan Salakhutdinov. A simple approach for visual room rearrangement: 3d mapping and semantic search. In *The Eleventh International Conference on Learning Representations*, 2022.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv: Arxiv-2305.16291*, 2023a.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *arXiv preprint arXiv: 2311.05997*, 2023b.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2021.

Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual room rearrangement. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021.

Erik Wijmans, Abhishek Kadian, Ari S. Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2020.

Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6750–6759, 2019.

Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models. *arXiv preprint arXiv:2305.05658*, 2023.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers, 2022. URL https://arxiv.org/abs/2203.08913.

Jingkang Yang, Yuhao Dong, Shuai Liu, Bo Li, Ziyue Wang, Chencheng Jiang, Haoran Tan, Jiamu Kang, Yuanhan Zhang, Kaiyang Zhou, et al. Octopus: Embodied vision-language programmer from environmental feedback. *arXiv preprint arXiv:2310.08588*, 2023.

Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. In *Proceedings of (ICLR) International Conference on Learning Representations*, May 2019.

Sriram Yenamandra, Arun Ramachandran, Karmesh Yadav, Austin Wang, Mukul Khanna, Theophile Gervet, Tsung-Yen Yang, Vidhi Jain, Alexander William Clegg, John Turner, et al. Homerobot: Open-vocabulary mobile manipulation. *arXiv preprint arXiv:2306.11565*, 2023.

Yichi Zhang and Joyce Chai. Hierarchical task learning from language instructions with unified transformers and self-monitoring. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 4202–4213, 2021.

Yichi Zhang, Jianing Yang, Jiayi Pan, Shane Storks, Nikhil Devraj, Ziqiao Ma, Keunwoo Yu, Yuwei Bao, and Joyce Chai. Danli: Deliberative agent for following natural language instructions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 1280–1298, 2022.

Kaizhi Zheng, Kaiwen Zhou, Jing Gu, Yue Fan, Jialu Wang, Zonglin Li, Xuehai He, and Xin Eric Wang. Jarvis: A neuro-symbolic commonsense reasoning framework for conversational embodied agents. 2022.

Hao Zhu, Raghav Kapoor, So Yeon Min, Winson Han, Jiatai Li, Kaiwen Geng, Graham Neubig, Yonatan Bisk, Aniruddha Kembhavi, and Luca Weihs. Excalibur: Encouraging and evaluating embodied exploration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14931–14942, 2023.

S1 Limitations

Our model has the following limitations:

1. Task planning from multimodal input: Currently, our LLM receives the environment's state only in case of a failure, through VLM feedback. Integrating the visual state of the environment in a more direct way may dramatically increase the accuracy of predicted plans. This direction aligns with recent work (Wang, 2023b; Mu, 2023; Yang, 2023) that uses visual features as input to language models.

2. Cost of GPT-4: While GPT-4 is the most accurate Large Language Model, its high cost necessitates exploring alternatives such as open-source models, hardware optimization, model compression or distillation of its knowledge to smaller models, to reduce expenses.

3. Manual Addition of Domains: Our model supports four domains with shared examples and prompts, but manual intervention is needed for adding significantly different domains and tasks. Future developments should focus on automating the detection and integration of out-of-domain inputs.

S2 Prompts

S2.1 Prompts for each module

We provide our full API (Listing S2), corrective API (Listing S3), prompt (Listing S4), re-planning prompt (Listing S5), and $Locator$ prompt (Listing S6).

S2.2 Prompt templates for prompt retrieval

In the prompt retrieval experiments, we include four prompt templates to be retrieved. These templates are shown for TEACh, ALFRED, Dialfred, and the Tidy Task in Listing S11, Listing S12, Listing S13, Listing S14, respectively.

S2.3 In-Context Examples

Samples of the in-context examples are shown for TEACh, ALFRED, Dialfred, and the Tidy Task in Listing S15, Listing S16, Listing S17, Listing S18, respectively.

S3 Question Asking

S3.1 Overview

In the DialFRED benchmark, when HELPER-X is unable to find an object, it is able to ask one of three question types in order to aid itself. In a real-world scenario, HELPER-X could take advantage of the LLM's capability to ask many types of questions, but the DialFRED benchmark limits us to three: direction, location, and appearance.

S3.2 Question asking pipeline

When HELPER-X does not have an object's location already in its memory or multiple instances of an objects exist in the memory, it forms a prompt with its current context and the API of available questions, as in Listing S19. Based on the context, HELPER-X then chooses and asks the most appropriate question. The returned answer and an API of search related actions, alongside the context and question, are then formed into another prompt, seen in Listing S20. Finally, this prompt is parsed by HELPER-X into an action script to search for the object. Examples of this full pipeline for if an object does not exist in the memory are in Listing S21 and Listing S22.

S4 Pre-conditions

An example of a pre-condition check for a macro-action is provided in Listing S7.

S5 Example LLM inputs & Outputs

We provide examples of dialogue input, retrieved examples, and LLM output for a TEACh sample in Listing S8, Listing S9, and Listing S10.

S6 Simulation environment

The TEACh dataset builds on the Ai2thor simulation environment (Kolve, 2017). At each time step the agent may choose from the following actions: Forward(), Backward(), Turn Left(), Turn Right(), Look Up(), Look Down(), Strafe Left(), Strafe Right(), Pickup(X), Place(X), Open(X), Close(X), ToggleOn(X), ToggleOff(X), Slice(X), and Pour(X), where X refers an object specified via a relative coordinate (x, y) on the egocentric RGB frame. Navigation actions move the agent in discrete steps. We rotate in the yaw direction by 90 degrees, and rotate in the pitch direction by 30 degrees. The RGB and depth sensors are at a resolution of 480x480, a field of view of 90 degrees, and lie at a height of 0.9015 meters. The agent's coordinates are parameterized by a single (x, y, z) coordinate triplet with x and z corresponding to movement in the horizontal plane and y reserved for the vertical direction. The TEACh benchmark allows a maximum of 1000 steps and 30 API failures per episode.

S7 $Executor$ details

S7.1 Semantic mapping and planning

Obstacle map HELPER-X maintains a 2D overhead occupancy map of its environment $\in \mathbb{R}^{H \times W}$ that it updates at each time step from the input RGB-D stream. The map is used for exploration and navigation in the environment.

At every time step t, we unproject the input depth maps using intrinsic and extrinsic information of the camera to obtain a 3D occupancy map registered to the coordinate frame of the agent, similar to earlier navigation agents Chaplot (2019). The 2D overhead maps of obstacles and free space are computed by projecting the 3D occupancy along the height direction at multiple height levels and summing. For each input RGB image, we run a SOLQ object segmentor (Dong, 2021) (pretrained on COCO Lin (2014) then finetuned on TEACh rooms) to localize each of 116 semantic object categories. For failure detection, we use a simple matching approach from Min (2021) to compare RGB pixel values before and after taking an action.

Object location and state tracking We maintain an object memory as a list of object detection 3D centroids and their predicted semantic labels $\{[(X, Y, Z)_i, \ell_i \in \{1...N\}], i = 1..K\}$, where K is the number of objects detected thus far. The object centroids are expressed with respect to the coordinate system of the agent, and, similar to the semantic maps, updated over time using egomotion. We track previously detected objects by their 3D centroid $C \in \mathbb{R}^3$. We estimate the centroid by taking the 3D point corresponding to the median depth within the segmentation mask and bring it to a common coordinate frame. We do a simple form of non-maximum suppression on the object memory, by comparing the euclidean distance of centroids in the memory to new detected centroids of the same category, and keep the one with the highest score if they fall within a distance threshold.

For each object in the object memory, we maintain an object state dictionary with a pre-defined list of attributes. These attributes include: category label, centroid location, holding, detection score, can use, sliced, toasted, clean, cooked. For the binary attributes, these are initialized by sending the object crop, defined by the detector mask, to the VLM model, and checking its match to each of [f"The {object_category} is {attribute}", f"The {object_category} is not {attribute}"]. We found that initializing these attributes with the VLM gave only a marginal difference to initializing them to default values in the TEACh benchmark, so we do not use it for the TEACh evaluations. However, we anticipate a general method beyond dataset biases of TEACh would much benefit from such vision-based attribute classification.

Table S1: **Alternative TEACh Execution from Dialog History (EDH) evaluation split.** Trajectory length weighted metrics are included in (parentheses). SR = success rate. GC = goal condition success rate. Note that Test Seen and Unseen are not the true TEACh test sets, but an alternative split of the validation set used until the true test evaluation is released, as mentioned in the TEACh github README, and also reported by DANLI (Zhang, 2022).

	Validation				**Test**			
	Unseen		**Seen**		**Unseen**		**Seen**	
	SR	GC	SR	GC	SR	GC	SR	GC
E.T.	8.35 (0.86)	6.34 (3.69)	8.28 (1.13)	8.72 (3.82)	7.38 (0.97)	6.06 (3.17)	8.82 (0.29)	9.46 (3.03)
DANLI	**17.25** (7.16)	23.88 (19.38)	16.89 (9.12)	25.10 (22.56)	16.71 (7.33)	23.00 (20.55)	**18.63** (9.41)	24.77 (21.90)
HELPER	**17.25** (3.22)	**25.24** (8.12)	**19.21** (4.72)	**33.54** (10.95)	**17.55** (2.59)	**26.49** (7.67)	17.97 (3.44)	**30.81** (8.93)

Exploration and path planning $HELPER-X$ explores the scene using a classical mapping method. We take the initial position of the agent to be the center coordinate in the map. We rotate the agent in-place and use the observations to instantiate an initial map. Second, the agent incrementally completes the maps by randomly sampling an unexplored, traversible location based on the 2D occupancy map built so far, and then navigates to the sampled location, accumulating the new information into the maps at each time step. The number of observations collected at each point in the 2D occupancy map is thresholded to determine whether a given map location is explored or not. Unexplored positions are sampled until the environment has been fully explored, meaning that the number of unexplored points is fewer than a predefined threshold.

To navigate to a goal location, we compute the geodesic distance to the goal from all map locations using graph search Inoue (2022) given the top-down occupancy map and the goal location in the map. We then simulate action sequences and greedily take the action sequence which results in the largest reduction in geodesic distance.

S7.2 2D-TO-3D UNPROJECTION

For the i-th view, a 2D pixel coordinate (u, v) with depth z is unprojected and transformed to its coordinate $(X, Y, Z)^T$ in the reference frame:

$$(X, Y, Z, 1) = \mathbf{G}_i^{-1} \left(z\frac{u - c_x}{f_x}, z\frac{v - c_y}{f_y}, z, 1 \right)^T \tag{1}$$

where (f_x, f_y) and (c_x, c_y) are the focal lengths and center of the pinhole camera model and $\mathbf{G}_i \in SE(3)$ is the camera pose for view i relative to the reference view. This module unprojects each depth image $I_i \in \mathbb{R}^{H\times W\times 3}$ into a pointcloud in the reference frame $P_i \in \mathbb{R}^{M_i\times 3}$ with M_i being the number of pixels with an associated depth value.

S8 ADDITIONAL DETAILS OF THE TIDY TASK

S8.1 METRIC DEFINITIONS IN THE TIDY TASK

The metrics in the original TIDEE paper (Sarch, 2022) require separate human evaluations on Amazon Mechanical Turk. We define a new set of metrics that does not require expensive annotations from humans for every evaluation. Below are detailed descriptions of each of the new metrics:

1. **Correctly Moved (CM)** Average number of correctly moved objects that are out of place in the scene, and moved by the agent. Higher is better.
2. **Incorrectly Moved (IM)** Average number of incorrectly moved objects that are not out of place, but were moved by the agent. Lower is better.
3. **Energy** Following ProcThor (Deitke, 2022), for each receptacle object, the probability that each object type appears on its surface is computed across the AI2THOR scenes. Here, we compute the total number of times each object type is on the receptacle type and divide it by the total number of times the receptacle type appears across the scenes. The energy metric in the Tidy Task is defined as follows:

$$(P_{cleanup} - P_{original})/(P_{dirty} - P_{original}) \tag{2}$$

where $P_{cleanup}$, P_{dirty}, and $P_{original}$ represent the sum of the object location probabilities for the cleaned up state of the room, the dirty/messy state of the room, and the original state of the room with objects put in-place by human designers, respectively. Lower is better.

4. **Steps** Average number of steps taken by the agent per episode.

S8.2 LANGAUGE INSTRUCTIONS FOR THE TIDY TASK

Since the Tidy Task does not include natural language instruction annotations, we formulate the language instruction as the following to give to the HELPER baseline and HELPER-X: *"Tidy up the house. These are the out of place objects: {detected_out_of_place_objects}. These are the receptacles in the current scene: {detected_receptacles}"*, where {detected_out_of_place_objects} are the objects classified as out of place, and {detected_receptacles} are any receptacle detected in the scene by the agent.

To obtain the list of out of place objects, we allow the agents use of the TIDEE (Sarch, 2022) visual detector to determine whether each object detected during the mapping phase is out of place. We found that out of place detection benefits significantly from visual detection in the Tidy Task, and thus we do not use an LLM for detecting the out of place attribute. Notably, adding the additional out of place attribute to the objects in the object memory can be shared across all benchmarks.

Listing S1: Full list of user personalization requests for the user personalization evaluation.

```
original input to LLM:
[['Driver', 'What is my task?'], ['Commander', "Make me a sandwich. The name of this
    sandwich is called the Larry sandwich. The sandwich has two slices of toast, 3
    slices of tomato, and 3 slice of lettuce on a clean plate."]]
[['Driver', 'What is my task?'], ['Commander', 'Make me a salad. The name of this salad
    is called the David salad. The salad has two slices of tomato and three slices of
    lettuce on a clean plate.']]
[['Driver', 'What is my task?'], ['Commander', "Make me a salad. The name of this salad
    is called the Dax salad. The salad has two slices of cooked potato. You'll need to
    cook the potato on the stove. The salad also has a slice of lettuce and a slice of
    tomato. Put all components on a clean plate."]]
[['Driver', 'What is my task?'], ['Commander', 'Make me breakfast. The name of this
    breakfast is called the Mary breakfast. The breakfast has a mug of coffee, and two
    slices of toast on a clean plate.']]
[['Driver', 'What is my task?'], ['Commander', 'Make me breakfast. The name of this
    breakfast is called the Lion breakfast. The breakfast has a mug of coffee, and
    four slices of tomato on a clean plate.']]
[['Driver', 'What is my task?'], ['Commander', 'Rearrange some objects. The name of
    this rearrangement is called the Lax rearrangement. Place three pillows on the
    sofa.']]
[['Driver', 'What is my task?'], ['Commander', 'Rearrange some objects. The name of
    this rearrangement is called the Pax rearrangement. Place two pencils and two pens
    on the desk.']]
[['Driver', 'What is my task?'], ['Commander', 'Clean some objects. The name of this
    cleaning is called the Gax cleaning. Clean two plates and two cups.']]
[['Driver', 'What is my task?'], ['Commander', "Make me a sandwich. The name of this
    sandwich is called the Gabe sandwich. The sandwich has two slices of toast, 2
    slices of tomato, and 1 slice of lettuce on a clean plate."]]
[['Driver', 'What is my task?'], ['Commander', 'Clean some objects. The name of this
    cleaning is called the Kax cleaning. Clean a mug and a pan.']]

No change:
"Make me the Larry sandwich"
"Make me the David salad"
"Make me the Dax salad"
"Make me the Mary breakfast"
"Make me the Lion breakfast"
"Complete the Lax rearrangement"
"Complete the Pax rearrangement"
"Perform the Gax cleaning"
"Make me the Gabe sandwich"
"Perform the Kax cleaning"

One change:
"Make me the Larry sandwich with four slices of lettuce"
"Make me the David salad with a slice of potato"
"Make me the Dax salad without lettuce"
"Make me the Mary breakfast with no coffee"
"Make me the Lion breakfast with three slice of tomato"
"Complete the Lax rearrangement with two pillows"
"Complete the Pax rearrangement but use one pencil instead of the the two pencils"
"Perform the Gax cleaning with three plates instead of two"
"Make me the Gabe sandwich with only 1 slice of tomato"
"Perform the Kax cleaning with only a mug"

Two changes:
"Make me the Larry sandwich with four slices of lettuce and two slices of tomato"
"Make me the David salad but add a slice of potato and add one slice of egg"
"Make me the Dax salad without lettuce and without potato"
"Make me the Mary breakfast with no coffee and add an egg"
"Make me the Lion breakfast with three slice of tomato and two mugs of coffee"
"Complete the Lax rearrangement with two pillows and add a remote"
```

```
"Complete the Pax rearrangement but use one pencil instead of the two pencils and add a
    book"
"Perform the Gax cleaning with three plates instead of the two plates and include a
    fork"
"Make me the Gabe sandwich with only 1 slice of tomato and two slices of lettuce"
"Perform the Kax cleaning without the pan and include a spoon"

Three changes:
"Make me the Larry sandwich with four slices of lettuce, two slices of tomato, and
    place all components directly on the countertop"
"Make me the David salad and add a slice of potato, add one slice of egg, and bring a
    fork with it"
"Make me the Dax salad without lettuce, without potato, and add an extra slice of
    tomato"
"Make me the Mary breakfast with no coffee, add an egg, and add a cup filled with water"
"Make me the Lion breakfast with three slice of tomato, two mugs of coffee, and add a
    fork"
"Complete the Lax rearrangement with two pillows, a remote, and place it on the arm
    chair instead"
"Complete the Pax rearrangement but use one pencil instead of the two pencils and
    include a book and a baseball bat"
"Perform the Gax cleaning with three plates instead of the two plates, include a fork,
    and do not clean any cups"
"Make me the Gabe sandwich with only 1 slice of tomato, two slices of lettuce, and add
    a slice of egg"
"Perform the Kax cleaning without the pan, include a spoon, and include a pot"
```

Listing S2: Full API for the parametrized macro-actions G used in the prompts.

```
class InteractionObject:
    """
    This class represents an expression that uniquely identifies an object in the house.
    """
    def __init__(self, object_class: str, landmark: str = None, attributes: list = []):
        '''
        object_class: object category of the interaction object (e.g., "Mug", "Apple")
        landmark: (optional if mentioned) landmark object category that the interaction
            object is in relation to (e.g., "CounterTop" for "apple is on the
            countertop")
        attributes: (optional) list of strings of desired attributes for the object.
            These are not necessarily attributes that currently exist, but ones that
            the object should eventually have. Attributes can only be from the
            following: "toasted", "clean", "cooked"
        '''
        self.object_class = object_class
        self.landmark = landmark
        self.attributes = attributes

    def pickup(self):
        """pickup the object.

        This function assumes the object is in view.

        Example:
        dialogue: <Commander> Go get the lettuce on the kitchen counter.
        Python script:
        target_lettuce = InteractionObject("Lettuce", landmark = "CounterTop")
        target_lettuce.go_to()
        target_lettuce.pickup()
        """
        pass

    def place(self, landmark_name):
        """put the interaction object on the landmark_name object.

        landmark_name must be a class InteractionObject instance

        This function assumes the robot has picked up an object and the landmark object
            is in view.

        Example:
        dialogue: <Commander> Put the lettuce on the kitchen counter.
        Python script:
        target_lettuce = InteractionObject("Lettuce", landmark = "CounterTop")
        target_lettuce.go_to()
        target_lettuce.pickup()
        target_countertop = InteractionObject("CounterTop")
        target_countertop.go_to()
        target_lettuce.place(target_countertop)
        """
        pass

    def slice(self):
        """slice the object into pieces.

        This function assumes the agent is holding a knife and the agent has navigated
            to the object using go_to().

        Example:
        dialogue: <Commander> Cut the apple on the kitchen counter.
        Python script:
```

```
        target_knife = InteractionObject("Knife") # first we need a knife to slice the
             apple with
        target_knife.go_to()
        target_knife.pickup()
        target_apple = InteractionObject("Apple", landmark = "CounterTop")
        target_apple.go_to()
        target_apple.slice()
        """
        pass

    def toggle_on(self):
        """toggles on the interaction object.

        This function assumes the interaction object is already off and the agent has
             navigated to the object.
        Only some landmark objects can be toggled on. Lamps, stoves, and microwaves are
             some examples of objects that can be toggled on.

        Example:
        dialogue: <Commander> Turn on the lamp.
        Python script:
        target_floorlamp = InteractionObject("FloorLamp")
        target_floorlamp.go_to()
        target_floorlamp.toggle_on()
        """
        pass

    def toggle_off(self):
        """toggles off the interaction object.

        This function assumes the interaction object is already on and the agent has
             navigated to the object.
        Only some objects can be toggled off. Lamps, stoves, and microwaves are some
             examples of objects that can be toggled off.

        Example:
        dialogue: <Commander> Turn off the lamp.
        Python script:
        target_floorlamp = InteractionObject("FloorLamp")
        target_floorlamp.go_to()
        target_floorlamp.toggle_off()
        """
        pass

    def go_to(self):
        """Navigate to the object

        """
        pass

    def open(self):
        """open the interaction object.

        This function assumes the landmark object is already closed and the agent has
             already navigated to the object.
        Only some objects can be opened. Fridges, cabinets, and drawers are some example
             of objects that can be closed.

        Example:
        dialogue: <Commander> Get the lettuce in the fridge.
        Python script:
        target_fridge = InteractionObject("Fridge")
        target_lettuce = InteractionObject("Lettuce", landmark = "Fridge")
        target_fridge.go_to()
        target_fridge.open()
```

```python
        target_lettuce.pickup()
        """
        pass

    def close(self):
        """close the interaction object.

        This function assumes the object is already open and the agent has already
            navigated to the object.
        Only some objects can be closed. Fridges, cabinets, and drawers are some example
            of objects that can be closed.
        """
        pass

    def clean(self):
        """wash the interaction object to clean it in the sink.

        This function assumes the object is already picked up.

        Example:
        dialogue: <Commander> Clean the bowl
        Python script:
        target_bowl = InteractionObject("Bowl", attributes = ["clean"])
        target_bowl.clean()
        """
        pass

    def put_down(self):
        """puts the interaction object currently in the agent's hand on the nearest
            available receptacle

        This function assumes the object is already picked up.
        This function is most often used when the holding object is no longer needed,
            and the agent needs to pick up another object
        """
        pass

    def pour(self, landmark_name):
        """pours the contents of the interaction object into the landmark object
            specified by the landmark_name argument

        landmark_name must be a class InteractionObject instance

        This function assumes the object is already picked up and the object is filled
            with liquid.
        """
        pass

    def fill_up(self):
        """fill up the interaction object with water

        This function assumes the object is already picked up. Note that only container
            objects can be filled with liquid.
        """
        pass

    def pickup_and_place(self, landmark_name):
        """go_to() and pickup() this interaction object, then go_to() and place() the
            interaction object on the landmark_name object.

        landmark_name must be a class InteractionObject instance
        """
        pass

    def empty(self):
```

```
        """Empty the object of any other objects on/in it to clear it out.

        Useful when the object is too full to place an object inside it.

        Example:
        dialogue: <Commander> Clear out the sink.
        Python script:
        target_sink = InteractionObject("Sink")
        target_sink.empty()
        """
        pass

    def cook(self):
        """Cook the object

        Example:
        dialogue: <Commander> Cook the potato.
        Python script:
        target_potato = InteractionObject("Potato", attributes = ["cooked"])
        target_potato.cook()
        """
        pass

    def toast(self):
        """Toast a bread slice in a toaster

        Toasting is only supported with slices of bread

        Example:
        dialogue: <Commander> Get me a toasted bread slice.
        Python script:
        target_breadslice = InteractionObject("BreadSliced", attributes = ["toasted"])
        target_breadslice.toast()
        """
        pass
```

Listing S3: Full Corrective API for the parametrized corrective macro-actions $G_{corrective}$ used in the prompts.

```
class AgentCorrective:
    '''
    This class represents agent corrective actions that can be taken to fix a subgoal
        error
    Example usage:
    agent = AgentCorrective()
    agent.move_back()
    '''

    def move_back(self):
        """Step backwards away from the object

        Useful when the object is too close for the agent to interact with it
        """
        pass

    def move_closer(self):
        """Step forward to towards the object to get closer to it

        Useful when the object is too far for the agent to interact with it
        """
        pass

    def move_alternate_viewpoint(self):
        """Move to an alternate viewpoint to look at the object

        Useful when the object is occluded or an interaction is failing due to collision
            or occlusion.
        """
        pass
```

Listing S4: Full Prompt for the . {} indicates areas that are replaced in the prompt.

```
You are an adept at translating human dialogues into sequences of actions for household
     robots. Given a dialogue between a <Driver> and a <Commander>, you convert the
     conversation into a Python program to be executed by a robot.

{API}

Write a script using Python and the InteractionObject class and functions defined above
     that could be executed by a household robot.

{RETRIEVED_EXAMPLES}

Adhere to these stringent guidelines:
1. Use only the classes and functions defined previously. Do not create functions that
     are not provided above.
2. Make sure that you output a consistent plan. For example, opening of the same object
     should not occur in successive steps.
3. Make sure the output is consistent with the proper affordances of objects. For
     example, a couch cannot be opened, so your output should never include the open()
     function for this object, but a fridge can be opened.
4. The input is dialogue between <Driver> and <Commander>. Interpret the dialogue into
     robot actions. Do not output any dialogue.
5. Object categories should only be chosen from the following classes: ShowerDoor,
     Cabinet, CounterTop, Sink, Towel, HandTowel, TowelHolder, SoapBar, ToiletPaper,
     ToiletPaperHanger, HandTowelHolder, SoapBottle, GarbageCan, Candle, ScrubBrush,
     Plunger, SinkBasin, Cloth, SprayBottle, Toilet, Faucet, ShowerHead, Box, Bed,
     Book, DeskLamp, BasketBall, Pen, Pillow, Pencil, CellPhone, KeyChain, Painting,
     CreditCard, AlarmClock, CD, Laptop, Drawer, SideTable, Chair, Blinds, Desk,
     Curtains, Dresser, Watch, Television, WateringCan, Newspaper, FloorLamp,
     RemoteControl, HousePlant, Statue, Ottoman, ArmChair, Sofa, DogBed, BaseballBat,
     TennisRacket, VacuumCleaner, Mug, ShelvingUnit, Shelf, StoveBurner, Apple,
     Lettuce, Bottle, Egg, Microwave, CoffeeMachine, Fork, Fridge, WineBottle, Spatula,
     Bread, Tomato, Pan, Cup, Pot, SaltShaker, Potato, PepperShaker, ButterKnife,
     StoveKnob, Toaster, DishSponge, Spoon, Plate, Knife, DiningTable, Bowl,
     LaundryHamper, Vase, Stool, CoffeeTable, Poster, Bathtub, TissueBox, Footstool,
     BathtubBasin, ShowerCurtain, TVStand, Boots, RoomDecor, PaperTowelRoll, Ladle,
     Kettle, Safe, GarbageBag, TeddyBear, TableTopDecor, Dumbbell, Desktop,
     AluminumFoil, Window, LightSwitch, AppleSliced, BreadSliced, LettuceSliced,
     PotatoSliced, TomatoSliced
6. You can only pick up one object at a time. If the agent is holding an object, the
     agent should place or put down the object before attempting to pick up a second
     object.
7. Each object instance should instantiate a different InteractionObject class even if
     two object instances are the same object category.
Follow the output format provided earlier. Think step by step to carry out the
     instruction.

Write a Python script that could be executed by a household robot for the following:
dialogue: {command}
Python script:
```

Listing S5: Full Prompt for the . {} indicates areas that are replaced in the prompt.

```
You are an excellent interpreter of human instructions for household tasks. Given a
     failed action subgoal by a household robot, dialogue instructions between robot
     <Driver> and user <Commander>, and information about the environment and failure,
     you provide a sequence of robotic subgoal actions to overcome the failure.

{API}

{API_CORRECTIVE}

Information about the failure and environment are given as follows:
Failed subgoal: The robotic subgoal for which the failure occured.
Execution error: feedback as to why the failed subgoal occurred.
Input dialogue: full dialogue instructions between robot <Driver> and user <Commander>
     for the complete task. This may or may not be useful.

I will give you examples of the input and output you will generate.
{retrieved_plans}

Fix the subgoal exectuion error using only the InteractionObject class and functions
     defined above that could be executed by a household robot. Follow these rules very
     strictly:
1. Important! Use only the classes and functions defined previously. Do not create
     functions or additional code that are not provided in the above API. Do not
     include if-else statements.
2. Important! Make sure that you output a consistent plan. For example, opening of the
     same object should not occur in successive steps.
3. Important! Make sure the output is consistent with the proper affordances of
     objects. For example, a couch cannot be opened, so your output should never
     include the open() function for this object, but a fridge can be opened.
4. Important! The dialogue is between <Driver> and <Commander>. The dialogue may or may
     not be helpful. Do not output any dialogue.
5. Important! Object classes should only be chosen from the following classes:
     ShowerDoor, Cabinet, CounterTop, Sink, Towel, HandTowel, TowelHolder, SoapBar,
     ToiletPaper, ToiletPaperHanger, HandTowelHolder, SoapBottle, GarbageCan, Candle,
     ScrubBrush, Plunger, SinkBasin, Cloth, SprayBottle, Toilet, Faucet, ShowerHead,
     Box, Bed, Book, DeskLamp, BasketBall, Pen, Pillow, Pencil, CellPhone, KeyChain,
     Painting, CreditCard, AlarmClock, CD, Laptop, Drawer, SideTable, Chair, Blinds,
     Desk, Curtains, Dresser, Watch, Television, WateringCan, Newspaper, FloorLamp,
     RemoteControl, HousePlant, Statue, Ottoman, ArmChair, Sofa, DogBed, BaseballBat,
     TennisRacket, VacuumCleaner, Mug, ShelvingUnit, Shelf, StoveBurner, Apple,
     Lettuce, Bottle, Egg, Microwave, CoffeeMachine, Fork, Fridge, WineBottle, Spatula,
     Bread, Tomato, Pan, Cup, Pot, SaltShaker, Potato, PepperShaker, ButterKnife,
     StoveKnob, Toaster, DishSponge, Spoon, Plate, Knife, DiningTable, Bowl,
     LaundryHamper, Vase, Stool, CoffeeTable, Poster, Bathtub, TissueBox, Footstool,
     BathtubBasin, ShowerCurtain, TVStand, Boots, RoomDecor, PaperTowelRoll, Ladle,
     Kettle, Safe, GarbageBag, TeddyBear, TableTopDecor, Dumbbell, Desktop,
     AluminumFoil, Window, LightSwitch, AppleSliced, BreadSliced, LettuceSliced,
     PotatoSliced, TomatoSliced
6. Important! You can only pick up one object at a time. If the agent is holding an
     object, the agent should place or put down the object before attempting to pick up
     a second object.
7. Important! Each object instance should instantiate a different InteractionObject
     class even if two object instances are the same object category.
8. Important! Your plan should ONLY fix the failed subgoal. Do not include plans for
     other parts of the dialogue or future plan that are irrelevant to the execution
     error and failed subgoal.
9. Important! output "do_nothing()" if the agent should not take any corrective actions.
Adhere to the output format I defined above. Think step by step to carry out the
     instruction.

Make use of the following information to help you fix the failed subgoal:
Failed subgoal: ...
Execution error: ...
```

```
Input dialogue: ...

You should respond in the following format:
Explain: Are there any steps missing to complete the subgoal? Why did the failed
    subgoal occur? What does the execution error imply for how to fix your future plan?
Plan (Python script): A Python script to only fix the execution error.

Explain:
```

Listing S6: Full Prompt for the $Locator$. {} indicates areas that are replaced in the prompt.

```
You are a household robot trying to locate objects within a house.
You will be given a target object category, your task is to output the top 3 most
     likely object categories that the target object category is likely to be found
     near: {OBJECT_CLASSES}
For your answer, take into account commonsense co-occurances of objects within a house
     and (if relevant) any hints given by the instruction dialogue between the robot
     <Driver> and user <Commander>.

For example, if given the target object category is "Knife" and the following dialogue:
     "<Commander> hi, make a slice of tomato. <Driver> where is the tomato? <Driver>
     where is the knife? <Commander> in the sink.", you might output the following top
     3 most likely object categories: "answer: Sink, CounterTop, Cabinet". Important:
     Your output should ONLY a list (3 words seperated by commas) of three object
     categories from the list above.

What are the top 3 most likely object categories for where to find the target category
     {INPUT_TARGET_OBJECT} near? Here is the dialogue that may or may not contain
     relevant information for your answer: {INPUT_DIALOGUE}

answer:
```

Listing S7: Example of a precondition check for the slice macro-action

```
if subgoal=="Slice":
   # contraint check for "slice" macro-action
   if object_tracker.get_label_of_holding()!="Knife":
      # first need to be holding a knife to slice
      program_to_add = [
         'target_knife = InteractionObject("Knife")',
         'target_knife.go_to()',
         'target_knife.pickup()',
      ]
```

Listing S8: Example of dialogue input, retrieved examples, and LLM output for a TEACh sample

```
Dialogue input:
<Driver> how can I help you today?<Commander> can you please make me a salad on a clean
    plate with tomato and cooked potato?<Driver> does the salad require chopped
    lettuce?<Commander> nope!<Driver> is that all?<Commander> can you place them on a
    plate?<Driver> are they not already on a plate?


Retrieved Examples:
Example 1:
dialogue: <Driver> What should I do today? <Commander> hi, make a slice of tomato.
    <Driver> where is the tomato? <Driver> where is the knife? <Commander> in the sink.
    <Driver> Tomato sliced. What next? <Commander> slice the potato. <Driver> Where is
    the potato? <Commander> in the microwave. <Commander> place all salad components on
    a plate. <Driver> How many slices of potato? <Commander> all salad components need
    to be place on a plate. <Driver> Where is the plate? <Commander> plate. <Commander>
    try the tomato piece. <Driver> done. <Commander> we are finished.
Python script:
target_tomato = InteractionObject("Tomato")
target_knife = InteractionObject("Knife") # need knife to slice tomato
target_knife.go_to() # first get knife
target_knife.pickup()
target_tomato.go_to()
target_tomato.slice()
target_tomato_slice1 = InteractionObject("TomatoSliced") # instantiate a tomato slice
    object after slicing tomato
target_potato = InteractionObject("Potato", landmark = "Microwave")
target_potato.go_to()
target_potato.slice()
target_potato_slice1 = InteractionObject("PotatoSliced") # instantiate a potato slice
    object after slicing potato
target_knife.put_down() # put down the knife to free up the agent's hand
target_plate = InteractionObject("Plate")
target_tomato_slice1.pickup_and_place(target_plate) # pick up the tomato slice and
    place it on the plate
target_potato_slice1.pickup_and_place(target_plate) # pick up the potato slice and
    place it on the plate

Example 2:
dialogue: <Driver> how can I help? <Commander> please serve 4 slices of tomato on a plate.
    <Driver> sure. <Driver> where can i find the tomato? <Driver> an knife. <Commander>
    Tomato on countertop and knife in fridge. <Commander> there should be a plate on the
    right cupboard by the sink. <Driver> completed. <Commander> great work thanks.
Python script:
target_tomato = InteractionObject("Tomato")
target_knife = InteractionObject("Knife") # need knife to slice tomato
target_knife.go_to() # first get knife
target_knife.pickup()
target_tomato.go_to()
target_tomato.slice()
target_plate = InteractionObject("Plate", landmark = "Sink")
# place 4 slices of the cut tomato on the plate
target_tomato_slice1 = InteractionObject("TomatoSliced") # instantiate first slice of
    tomato
target_tomato_slice2 = InteractionObject("TomatoSliced") # instantiate second slice of
    tomato
target_tomato_slice3 = InteractionObject("TomatoSliced") # instantiate third slice of
    tomato
target_tomato_slice4 = InteractionObject("TomatoSliced") # instantiate fourth slice of
    tomato
target_tomato_slice1.pickup_and_place(target_plate) # pick up the first tomato slice
    and place it on the plate
target_tomato_slice2.pickup_and_place(target_plate) # pick up the second tomato slice
    and place it on the plate
target_tomato_slice3.pickup_and_place(target_plate) # pick up the third tomato slice
    and place it on the plate
```

```
target_tomato_slice4.pickup_and_place(target_plate) # pick up the fourth tomato slice
    and place it on the plate

Example 3:
dialogue: <Driver> hi, what is task. <Commander> make a plate of toast. <Commander>
    bread in the sink. <Driver> knife. <Commander> on the chair beside the table.
    <Commander> clean the plate. <Driver> there is a clean one, do i need to clean the
    other one. <Commander> you can use the clean one. <Driver> done. <Commander> good
    job. <Driver> :).
Python script:
target_bread = InteractionObject("Bread", landmark = "Sink")
target_knife = InteractionObject("Knife", landmark = "Chair")
target_toaster = InteractionObject("Toaster") # need knife to slice bread
target_knife.go_to() # first get knife
target_knife.pickup()
target_bread.go_to()
target_bread.slice() # slice bread using knife
target_knife.put_down() # put down the knife to free up the agent's hand
target_bread_slice1 = InteractionObject("BreadSliced") # instantiate a bread slice
    object after slicing bread
target_bread_slice1.pickup_and_place(target_toaster) # pick up the bread slice and
    place it in the toaster
target_toaster.toggle_on() # turn on toaster to toast bread
target_toaster.toggle_off() # turn off toaster to complete toasting process
# place toast on plate
target_plate = InteractionObject("Plate")
target_plate.clean()
target_plate.put_down()
target_bread_slice1.pickup_and_place(target_plate) # pick up the toasted bread slice
    and place it on the plate

LLM output:
target_tomato = InteractionObject("Tomato")
target_potato = InteractionObject("Potato")
target_plate = InteractionObject("Plate")
target_stove = InteractionObject("StoveBurner")

# Clean the plate
target_plate.go_to()
target_plate.clean()

# Cook the potato
target_potato.go_to()
target_potato.pickup()
target_stove.go_to()
target_potato.place(target_stove)
target_stove.toggle_on()
target_stove.toggle_off()

# Place the cooked potato on the plate
target_potato = InteractionObject("Potato")
target_potato.pickup()
target_plate.go_to()
target_potato.place(target_plate)

# Place the tomato on the plate
target_tomato.go_to()
target_tomato.pickup()
target_plate.go_to()
target_tomato.place(target_plate)
```

Listing S9: Example of dialogue input, retrieved examples, and LLM output for a TEACh sample

```
Dialogue input:
<Driver> what do i do today<Commander> Hi. Please place two tissue boxes on a table. One
     is on the gray chair in the corner.<Commander> The other is under the tv<Commander>
     tv*<Driver> where is the other one


Retrieved Examples:
Example 1:
dialogue: <Driver> hi, what is task. <Commander> make a plate of toast. <Commander>
     bread in the sink. <Driver> knife. <Commander> on the chair beside the table.
     <Commander> clean the plate. <Driver> there is a clean one, do i need to clean the
     other one. <Commander> you can use the clean one. <Driver> done. <Commander> good
     job. <Driver> :).
Python script:
target_bread = InteractionObject("Bread", landmark = "Sink")
target_knife = InteractionObject("Knife", landmark = "Chair")
target_toaster = InteractionObject("Toaster") # need knife to slice bread
target_knife.go_to() # first get knife
target_knife.pickup()
target_bread.go_to()
target_bread.slice() # slice bread using knife
target_knife.put_down() # put down the knife to free up the agent's hand
target_bread_slice1 = InteractionObject("BreadSliced") # instantiate a bread slice
     object after slicing bread
target_bread_slice1.pickup_and_place(target_toaster) # pick up the bread slice and
     place it in the toaster
target_toaster.toggle_on() # turn on toaster to toast bread
target_toaster.toggle_off() # turn off toaster to complete toasting process
# place toast on plate
target_plate = InteractionObject("Plate")
target_plate.clean()
target_plate.put_down()
target_bread_slice1.pickup_and_place(target_plate) # pick up the toasted bread slice
     and place it on the plate

Example 2:
dialogue: <Driver> how can I help? <Commander> please serve 4 slices of tomato on a plate.
     <Driver> sure. <Driver> where can i find the tomato? <Driver> an knife. <Commander>
     Tomato on countertop and knife in fridge. <Commander> there should be a plate on the
     right cupboard by the sink. <Driver> completed. <Commander> great work thanks.
Python script:
target_tomato = InteractionObject("Tomato", landmark = "CounterTop")
target_knife = InteractionObject("Knife", landmark = "Fridge") # need knife to slice
     tomato
target_knife.go_to() # first get knife
target_knife.pickup()
target_tomato.go_to()
target_tomato.slice()
target_plate = InteractionObject("Plate", landmark = "Sink")
# place 4 slices of the cut tomato on the plate
target_tomato_slice1 = InteractionObject("TomatoSliced") # instantiate first slice of
     tomato
target_tomato_slice2 = InteractionObject("TomatoSliced") # instantiate second slice of
     tomato
target_tomato_slice3 = InteractionObject("TomatoSliced") # instantiate third slice of
     tomato
target_tomato_slice4 = InteractionObject("TomatoSliced") # instantiate fourth slice of
     tomato
target_tomato_slice1.pickup_and_place(target_plate) # pick up the first tomato slice
     and place it on the plate
target_tomato_slice2.pickup_and_place(target_plate) # pick up the second tomato slice
     and place it on the plate
target_tomato_slice3.pickup_and_place(target_plate) # pick up the third tomato slice
     and place it on the plate
target_tomato_slice4.pickup_and_place(target_plate) # pick up the fourth tomato slice
     and place it on the plate
```

```
Example 3:
dialogue: <Driver> How can I help? <Commander> put all the scrub brushes on the counter
     top. <Driver> where can I find them? <Commander> one is on top of the toilet.
     <Driver> there is only a cloth. <Driver> I put it on the counter top. <Driver> are
     there more? <Commander> can you try with the brush to the side of the toilet please?
     <Driver> okay.
Python script:
target_scrub_brush1 = InteractionObject("ScrubBrush", landmark = "Toilet")
target_counter_top = InteractionObject("CounterTop")
target_scrub_brush1.pickup_and_place(target_counter_top) # pick up the first ScrubBrush
     and place it on the CounterTop
target_scrub_brush2 = InteractionObject("ScrubBrush", landmark = "Toilet")
target_scrub_brush2.pickup_and_place(target_counter_top) # pick up the second
     ScrubBrush and place it on the CounterTop

LLM output:
target_tissue_box1 = InteractionObject("TissueBox", landmark = "Chair")
target_table = InteractionObject("Table")
target_tissue_box1.pickup_and_place(target_table) # pick up the first TissueBox and
     place it on the Table
target_tissue_box2 = InteractionObject("TissueBox", landmark = "TVStand")
target_tissue_box2.pickup_and_place(target_table) # pick up the second TissueBox and
     place it on the Table
```

Listing S10: Example of dialogue input, retrieved examples, and LLM output for a TEACh sample

```
Dialogue input:
<Driver> hi how can i help<Commander> Can you prepare me a coffee in a clean
    mug<Commander> There should be a mug on the white table<Commander> it might need
    cleaning first<Commander> that's great<Driver> are they not already on a plate?


Retrieved Examples:
Example 1:
dialogue: <Commander> Prepare coffee in a clean mug.
Python script:
target_mug = InteractionObject("Mug")
target_mug.go_to()
target_mug.pickup()
# clean mug in the sink
target_mug.clean()
target_coffee_machine = InteractionObject("CoffeeMachine")
target_coffee_machine.go_to()
target_mug.place(target_coffee_machine)
target_coffee_machine.toggle_on() # turn on coffee machine to make coffee
target_coffee_machine.toggle_off() # turn off coffee machine after done

Example 2:
dialogue: <Driver> what can i for you today? <Commander> could you boil a potato?
    <Driver> sure thing! <Driver> are there any pots? <Commander> pots for boiling
    potato? <Driver> yes. <Commander> just the one that you were holding earlier and a
    couple others of the same size. <Commander> does it not fit in the sink? <Driver> no.
    <Commander> see that mug of water next to the pot? <Driver> yes. <Commander> pour
    the mugs water into the pot. <Driver> thanks. <Commander> you're welcome!
    <Commander> excellent thank you!
Python script:
target_pot = InteractionObject("Pot")
target_mug = InteractionObject("Mug", landmark = "Pot")
target_mug.go_to()
target_mug.pickup()
target_mug.pour(target_pot)
target_mug.put_down() # put down mug on any location to free up agent's hand
target_potato = InteractionObject("Potato")
target_potato.pickup_and_place(target_pot) # pick up the potato and place it in the pot
target_stove_knob = InteractionObject("StoveKnob") # turn on the stove to boil potato
    by turning on the stove
target_stove_knob.go_to()
target_stove_knob.toggle_on() # boil water
target_stove_knob.toggle_off() # complete boiling process by turning off

Example 3:
dialogue: <Driver> hi, what is task. <Commander> make a plate of toast. <Commander>
    bread in the sink. <Driver> knife. <Commander> on the chair beside the table.
    <Commander> clean the plate. <Driver> there is a clean one, do i need to clean the
    other one. <Commander> you can use the clean one. <Driver> done. <Commander> good
    job. <Driver> :).
Python script:
target_bread = InteractionObject("Bread", landmark = "Sink")
target_knife = InteractionObject("Knife", landmark = "Chair")
target_toaster = InteractionObject("Toaster", "Toaster") # need knife to slice bread
target_knife.go_to() # first get knife
target_knife.pickup()
target_bread.go_to()
target_bread.slice() # slice bread using knife
target_knife.put_down() # put down the knife to free up the agent's hand
target_bread_slice1 = InteractionObject("BreadSliced") # instantiate a bread slice
    object after slicing bread
target_bread_slice1.pickup_and_place(target_toaster) # pick up the bread slice and
    place it in the toaster
target_toaster.toggle_on() # turn on toaster to toast bread
target_toaster.toggle_off() # turn off toaster to complete toasting process
# place toast on plate
```

```
target_plate = InteractionObject("Plate")
target_plate.clean()
target_plate.put_down()
target_bread_slice1.pickup_and_place(target_plate) # pick up the toasted bread slice
    and place it on the plate


LLM output:
target_table = InteractionObject("Table")
target_mug = InteractionObject("Mug", landmark = "Table")
target_mug.go_to()
target_mug.pickup()
# clean mug in the sink
target_sink = InteractionObject("Sink")
target_mug.place(target_sink)
target_mug.clean()
target_mug.pickup()
target_coffee_machine = InteractionObject("CoffeeMachine")
target_coffee_machine.go_to()
target_mug.place(target_coffee_machine)
target_coffee_machine.toggle_on() # turn on coffee machine to make coffee
target_coffee_machine.toggle_off() # turn off coffee machine after done
```

Listing S11: Prompt template for TEACh

```
You are an adept at translating human dialogues into sequences of actions for household
     robots. Given a dialogue between a <Driver> and a <Commander>, you convert the
     conversation into a Python program to be executed by a robot.

{API}

Write a script using Python and the InteractionObject class and functions defined above
     that could be executed by a household robot.

Here are a few examples of typical inputs and outputs (only for in-context reference):
{RETRIEVED_EXAMPLES}

Adhere to these stringent guidelines:
1. Use only the classes and functions defined previously. Do not create functions that
     are not provided above.
2. Make sure that you output a consistent plan. For example, opening of the same object
     should not occur in successive steps.
3. Make sure the output is consistent with the proper affordances of objects. For
     example, a couch cannot be opened, so your output should never include the open()
     function for this object, but a fridge can be opened.
4. The input is dialogue between <Driver> and <Commander>. Interpret the dialogue into
     robot actions. Do not output any dialogue.
5. Object categories should only be chosen from the following classes: {OBJECT_CLASSES}
6. You can only pick up one object at a time. If the agent is holding an object, the
     agent should place or put down the object before attempting to pick up a second
     object.
7. Each object instance should instantiate a different InteractionObject class even if
     two object instances are the same object category.
Follow the output format provided earlier. Think step by step to carry out the
     instruction.

Write a Python script that could be executed by a household robot for the following:
dialogue: {command}
Python script:
```

Listing S12: Prompt template for ALFRED

```
You are an excellent interpreter of instructions for household tasks. Given a task
    overview <High Level Goal> and step to perform <Low Level Goal>, you break the
    instructions down into a sequence of robotic actions.

{API}

Write a script using Python and the InteractionObject class and functions defined above
    that could be executed by a household robot.

Here are a few examples of typical inputs and outputs (only for in-context reference):
{RETRIEVED_EXAMPLES}

Adhere to these stringent guidelines:
1. Use only the classes and functions defined previously. Do not create functions that
    are not provided above.
2. Make sure that you output a consistent plan. For example, opening of the same object
    should not occur in successive steps.
3. Make sure the output is consistent with the proper affordances of objects. For
    example, a couch cannot be opened, so your output should never include the open()
    function for this object, but a fridge can be opened.
4. The input is high level task description and low level subgoals to perform the high
    level task. Interpret the instructions into robot actions.
5. Object categories should only be chosen from the following classes: {OBJECT_CLASSES}
6. You can only pick up one object at a time. If the agent is holding an object, the
    agent should place or put down the object before attempting to pick up a second
    object.
7. Each object instance should instantiate a different InteractionObject class even if
    two object instances are the same object category.
8. Always focus on solving the high level goal. Low level instructions should only be
    used to guide and plan better.
9. Before performing each action, check if that action is allowed for a particular
    receptacle class. A few examples have been given in API documentation.
10. Check if the receptacle needs to be opened before placing the object. If yes, then
    open the receptacle before placing the object.
Follow the output format provided earlier. Think step by step to carry out the
    instruction.

Write a Python script that could be executed by a household robot for the following:
{command}
Python script:
```

Listing S13: Prompt template for the Dialfred

```
You are an excellent interpreter of instructions for household tasks. Given a task
    overview <High Level Goal> and step to perform <Low Level Goal>, you break the
    instructions down into a sequence of robotic actions.

{API}

Write a script using Python and the InteractionObject class and functions defined above
    that could be executed by a household robot.

Here are a few examples of typical inputs and outputs (only for in-context reference):
{RETRIEVED_EXAMPLES}

Adhere to these stringent guidelines:
1. Use only the classes and functions defined previously. Do not create functions that
    are not provided above.
2. Make sure that you output a consistent plan. For example, opening of the same object
    should not occur in successive steps.
3. Make sure the output is consistent with the proper affordances of objects. For
    example, a couch cannot be opened, so your output should never include the open()
    function for this object, but a fridge can be opened.
4. The input is high level task description and low level subgoals to perform the high
    level task. Interpret the instructions into robot actions.
5. Object categories should only be chosen from the following classes: {OBJECT_CLASSES}
6. You can only pick up one object at a time. If the agent is holding an object, the
    agent should place or put down the object before attempting to pick up a second
    object.
7. Each object instance should instantiate a different InteractionObject class even if
    two object instances are the same object category.
8. Make sure that you are solving both the high level goal and the low level goals.
    Some instructions may only be present in one or the other, so address everything
    from both.
9. Before performing each action, check if that action is allowed for a particular
    receptacle class. A few examples have been given in API documentation.
Follow the output format provided earlier. Think step by step to carry out the
    instruction.

Write a Python script that could be executed by a household robot for the following:
{command}
Python script:
```

Listing S14: Prompt template for the Tidy Task

```
Task: As an AI-driven cleaning robot, you are tasked with employing commonsense
     reasoning to identify where to place out of place objects that aren't situated
     appropriately. Given a list of out of place objects, you are to write a Python
     program to be executed by a robot that will bring the out of place objects to a
     suitable location.

{API}

Write a script using Python and the InteractionObject class and functions defined above
     that could be executed by a household robot.

Here are a few examples of typical inputs and outputs (only for in-context reference):
{RETRIEVED_EXAMPLES}

Adhere to these stringent guidelines:
1. Use only the classes and functions defined previously. Do not create functions that
     are not provided above.
2. Make sure that you output a consistent plan. For example, opening of the same object
     should not occur in successive steps.
3. Make sure the output is consistent with the proper affordances of objects. For
     example, a couch cannot be opened, so your output should never include the open()
     function for this object, but a fridge can be opened.
4. Object categories should only be chosen from the following classes: {OBJECT_CLASSES}
5. You can only pick up one object at a time. If the agent is holding an object, the
     agent should place or put down the object before attempting to pick up a second
     object.
6. Each object instance should instantiate a different InteractionObject class even if
     two object instances are the same object category.
7. Address each item systematically, one by one.
8. Base your decisions on your ingrained knowledge about the typical placement of
     day-to-day objects.
Follow the output format provided earlier. Think step by step to carry out the
     instruction.

Write a Python script that could be executed by a household robot for the following:
input: {command}
Python script:
```

Listing S15: Sample in-context example for TEACh

```
Dialogue input:

<Driver> what shall I do today? <Commander> clean all the bowls. <Driver> where are the
    bowls? <Commander> start with the one by the stove. <Commander> left. <Commander>
    rinse it with water. <Commander> great. <Driver> what next? <Commander> the next
    one is in the fridge. <Commander> you need to rinse it with water also.
    <Commander> great job. we are finished.

Python script:
target_bowl1 = InteractionObject("Bowl", landmark = "Stove", attributes = ["clean"])
target_bowl1.go_to()
target_bowl1.pickup()
target_bowl1.clean()
target_bowl1.put_down()
target_bowl2 = InteractionObject("Bowl", landmark = "Fridge", attributes = ["clean"])
target_bowl2.go_to()
target_bowl2.pickup()
target_bowl2.clean()
target_bowl2.put_down()
```

Listing S16: Sample in-context example for ALFRED

```
High Level Goal: To heat an apple and place in the black bin.
Low Level Goal:
(1) Turn around and walk to the kitchen island.
(2) Pick up the apple in front of the gold colored plate.
(3) Walk around the kitchen island and to the stove on the right, look above the stove to
face the microwave.
(4) Place the apple inside the microwave, heat up/cook the apple, take the apple out of
    the
microwave.
(5) Turn left, turn left at the fridge, turn left to face the kitchen island, and look
    down
at the black bin.
(6) Place the apple in the bin on the right side.
Python script:
target_apple = InteractionObject("Apple", landmark = "CounterTop")
target_apple.go_to()
target_apple.pickup()
target_microwave = InteractionObject("Microwave")
target_microwave.go_to()
target_microwave.open() # open microwave before placing
target_apple.place(target_microwave)
target_microwave.close() # close microwave before toggle on
target_microwave.toggle_on() # toggle on to heat up & cook
target_microwave.toggle_off() # Important! toggle off to end heating. Do not try to
    open microwave without toggle off!
target_microwave.open() # open microwave before picking
target_apple.pickup()
target_trashcan = InteractionObject("GarbageCan")
target_trashcan.go_to()
target_apple.place(target_trashcan)
```

Listing S17: Sample in-context example for Dialfred

```
High Level Goal: Put the watch in the bowl on the nightstand.
Low Level Goal:
(1) Turn around, walk around the bookshelf, turn to the desk.
(2) Pick the watch up off of the desk.
(3) Put the watch in the bowl on the desk.
(4) Pick up the bowl off of the desk.
(5) Walk back around the bookshelf, walk over between the two beds.
(6) Put the bowl on the nightstand.
Python script:
target_watch = InteractionObject("Watch", landmark = "Desk")
target_watch.go_to()
target_watch.pickup()
target_bowl = InteractionObject("Bowl", landmark = "Desk")
target_bowl.go_to()
target_watch.place(target_bowl)
target_bowl.go_to()
target_bowl.pickup()
target_nightstand = InteractionObject("SideTable", landmark = "Bed")
target_nightstand.go_to()
target_bowl.place(target_nightstand)
```

Listing S18: Sample in-context example for the Tidy Task

```
input: Tidy up the house. These are the out of place objects: Potato, Knife. These are
the receptacles in the current scene: DiningTable, Microwave, CoffeeMachine, CounterTop.
Python script:
# initialize the out of place objects
target_potato = InteractionObject("Potato")
target_knife = InteractionObject("Knife")
# initialize the placement objects to place the out of place object on
target_countertop = InteractionObject("CounterTop") # The best, commonsense location
    for both the potato and knife is on the countertop.
# re-position potato to the countertop to tidy it up
target_potato.go_to()
target_potato.pickup()
target_countertop.go_to()
target_potato.place(target_countertop)
# re-position knife to the countertop to tidy it up
target_knife.go_to()
target_knife.pickup()
target_countertop.go_to()
target_knife.place(target_countertop)
```

Listing S19: Prompt template for Question Selection

```
You are an excellent interpreter of human instructions for household tasks. Given a
     list of questions you can ask and information about the current environment and
     context, you provide a question that should be asked in order to give the agent
     useful information.

{API}

Write a script using Python using the class and functions defined above that could be
     executed by a household robot.

Adhere to these stringent guidelines:
1. Use only the classes and functions defined previously. Do not create functions that
     are not provided above.
2. Make sure you choose the question that provides the most information and is most
     relevant for the situation at hand.
3. Object categories should only be chosen from the following classes: {OBJECT_CLASSES}
Follow the output format provided earlier. Think step by step to carry out the
     instruction.

Write a Python script that asks questions to help a household robot in the following
     situation:
{context}
Python script:
```

Listing S20: Prompt template for Answer Parsing

```
You are an excellent interpreter of human instructions for household tasks. Given the
    current context of the agent, a question that was asked, and an answer that was
    given, you must write code for actions the agent should take based on the answer
    provided.

{API}

Write a script using Python using the class and functions defined above that could be
    executed by a household robot.

Adhere to these stringent guidelines:
1. Use only the classes and functions defined previously. Do not create functions that
    are not provided above.
2. Make sure you plan the most simple and direct interpretation of the answer given.
3. Prioritize the most specific information given. For example, an actual object name
    should be deemed more important than a region.
4. If multiple pieces of information are given, ensure you incorporate all of them into
    the script.
5. Object categories should only be chosen from the following classes: {OBJECT_CLASSES}

Write a Python script that asks questions to help a household robot in the following
    situation:
{context}
{question}
{answer}
Python script:
```

Listing S21: Sample in-context example 1 for Question Asking

```
Question Selection Input:
Context: The agent does not know where the ButterKnife is.
Questioning Script:
askForLocation('ButterKnife')

Answer Parsing Input:
Context: The agent does not know where the ButterKnife is.
Question Asked: askForLocation('ButterKnife')
Answer Returned: The ButterKnife is to your left on the countertop.

Parsed Answer:
# Turn to the left as per the instruction
turn('left')
# Search for the butterknife on the counter top
search_near_other_object('ButterKnife', 'CounterTop')
```

Listing S22: Sample in-context example 2 for Question Asking

```
Question Selection Input:
Context: The agent does not know where the SoapBar is.
Questioning Script:
askForLocation('SoapBar')

Answer Parsing Input:
Context: The agent does not know where the SoapBar is.
Question Asked: askForLocation('SoapBar')
Answer Returned: The SoapBar is to your front right in the garbage can.
Parsed Answer:
# Turn right as the SoapBar is to the front right
turn('right')
# Move forward to reach the garbage can
move('forward')
# Search for the SoapBar near the garbage can
search_near_other_object('SoapBar', 'GarbageCan')
```